NEURAL NETWORK METHODS FOR APPLICATION IN EDUCATIONAL MEASUREMENT

by

Geoffrey Converse

A thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Applied Mathematical and Computational Sciences
in the Graduate College of
The University of Iowa

August 2021

Thesis Committee: Suely Oliveira, Thesis Supervisor
Mariana Curi
David Stewart
Jonathan Templin
Colleen Mitchell

Copyright by
GEOFFREY CONVERSE
2021
All Rights Reserved

ACKNOWLEDGEMENTS

I have so many people to thank for their support during my pursuit of a Ph.D. at the University of Iowa. My advisor, Suely Oliveira, who was always willing to listen to my ideas, even when they weren't very good. I was given many opportunities to be a part of interesting projects and grow as a researcher and leader. Through Dr. Oliveira, I was introduced to Mariana Curi – I'm appreciative of the time that Dr. Curi spent with me (both in Iowa and virtually), and her willingness to work with me on the research that became my thesis. I also must thank the three other members of my committee, Drs. Jonathan Templin, David Stewart, and Colleen Mitchell for their mentorship, ideas, and guidance.

I had a great opportunity to participate in an internship at ACT, for which I am grateful. In particular, I want to thank Scott and Yuchi for their role in introducing me to new areas of machine learning. Their work and guidance laid the foundation for a second topic to be inserted into my thesis.

I want to thank the Mathematics and Computer Science departments for the challenging coursework and instruction which allowed me to grow academically, as well as the Graduate College, all of which provided financial support during my time at Iowa.

Stepping away from math, I also am endebted to my parents, Ken and Karna, for their unwavering encouragement and the abundance of opportunities that they made avaiable to me throughout life. I thank my other family and friends in Iowa

City, Des Moines, and elsewhere for their continuous moral support and friendship.

Lastly but certainly not least, I thank my wife Danielle, who has always been available to provide her love, reassurance, and support. She knows me better than anyone, and has stood beside me throughout all the stressful times of graduate school.

ABSTRACT

In educational measurement, Item Response Theory (IRT) provides a means of quantifying student knowledge. Specifically, IRT models the probability of a student answering a particular item correctly as a function of the student's continuous-valued latent abilities $\mathbf{\Theta}$ (e.g. add, subtract, multiply, divide) and parameters associated with the item (e.g. difficulty). Given a group of students' binary responses (correct/incorrect) to an assessment, parameter estimation techniques are used to infer the students' abilities $\mathbf{\Theta}$ to better evaluate student performance. But as the number of students, items, and dimension of $\mathbf{\Theta}$ increases, traditional parameter estimation methods which rely on numerical integration or MCMC techniques become infeasible.

In this thesis, a novel modification to a variational autoencoder (VAE), an unsupervised learning method, is presented which incorporates multiple pieces of domain knowledge into the VAE architecture. Specifically, an expert-annotated Q-matrix detailing the association between items and abilities is used to constrain the trainable weights in the VAE decoder. This directly links the generative posterior distribution of a VAE to IRT models, and allows interpretation of trainable weights as item parameters and a hidden neural layer as student ability estimates. The use of a neural network in this application allows for efficient estimation of paramaters in high-dimensional datasets.

An additional alteration to the VAE encoder allows for modeling of correlated, rather than independent, latent abilities. This architecture is also of interest to

areas outside of education where the probability distribution of the VAE latent code is known. The proposed method, titled ML2P-VAE, acheives accuracy similar to traditional parameter estimation methods on smaller datasets and provides quality estimates on educational assessments with a large number of students, items, and latent abilities, where traditional methods struggle.

Modifications used in ML2P-VAE are extended to integrate IRT into deep knowledge tracing models, which use time-dependent neural networks such as long short-term memory networks and Transformers. In an online environment where many items are available, the task of knowledge tracing is to track a student's learning dynamically as they progress through an assessment. This task has received more attention recently due to the emergence of online learning and AI tutoring systems, where following student's knowledge acquisition can help facilitate automated curriculum in real time. The inclusion of IRT in the knowledge tracing framework presents a trade-off between explainability and accuracy, while remaining competitive with state-of-the-art methods.

PUBLIC ABSTRACT

In this work, multiple neural network techniques are presented for use in educational measurement applications. In educational measurement, the goal is to quantify student learning based on assessment performance. Item Response Theory (IRT) models the probability of students answering individual questions correctly as a function of the particular student's set of latent abilities (e.g. add, subtract, multiply, divide), as well as parameters pertaining to the specific question (e.g. difficulty).

A common task is to measure a population of students' knowledge, given their responses to an assessment. Though many methods exist for this purpose, measurement becomes difficult or infeasible as the number of students, questions, and latent abilities increase. This issue has become more problematic in the age of Big Data, as electronic learning environments increase in functionality and popularity and educational assessments grow in scale and scope.

In this thesis, a novel neural network architecture is presented which estimates question and student parameters from high-dimensional assessment data with ease. The modifications made to existing deep learning methods provide a direct link to IRT and have clear use in applications such as AI tutoring systems. The proposed machine learning architecture is relevant from perspectives outside of education as well, as it introduces interpretability into otherwise black-box deep learning models.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

Figure

CHAPTER 1
INTRODUCTION

In educational measurement, the goal is to quantify student learning in order to better assess the needs of individuals. In particular, Item Response Theory (IRT) provides a means of modeling the probability of students answering items (i.e. questions) of an assessment correctly [57], dependent on both the student's latent ability (i.e. knowledge, skill) and the difficulty and relevance of the item. IRT quantifies a student's latent ability as a continuous value $\boldsymbol{\Theta}$. In this work, we focus on the multidimensional case where $\boldsymbol{\Theta}$ is a vector representing multiple skills required for a particular assessment [78], such as the skills "add," "subtract," "multiply," and "divide" on an elementary math exam.

In real applications, the latent trait $\boldsymbol{\Theta}$ is unobservable, and the task is to infer each student's knowledge level from their correct and incorrect responses on an assessment. Many methods exist for this task, focusing around maximum likelihood estimation [4, 16], the EM algorithm [11, 33], and Monte-Carlo methods [14, 70]. These are all popular parameter estimation techniques in computational statistics, yet experience difficulties when faced with high-dimensional large datasets. Specifically, when the number of students, items, and latent abilities increase, these traditional approaches become infeasible due to computational complexity.

Recent years have seen an upswing in the popularity of artificial neural networks in many fields, due to their strong predictive power and ability to learn deep contextual representations [64, 94]. But this high level of accuracy comes with the

price of explainability – most neural network models act as a black-box, where the decision-making process and understanding of parameters is not interpretable [102].

The primary contribution of this thesis is the development of an auto-encoding neural network approach to IRT parameter estimation which remains efficient on large datasets (a sizable hurdle of traditional IRT methods), while introducing explainability into the neural network (a drawback of deep learning models). The proposed method, titled "ML2P-VAE," modifies the architecture of a variational autoencoder (VAE) [51] using domain knowledge of experts in the educational field [25, 39] in order to convert the VAE decoder into an approximate IRT model, parameterized and learned by a neural network. Despite being an unsupervised learning method, ML2P-VAE allows for interpretation of a hidden neural layer as estimates to student's latent ability $\boldsymbol{\Theta}$.

The significance of the novel ML2P-VAE method is summarized as follows. The use of a VAE, rather than a regular autoencoder [39], greatly increases the accuracy of parameter estimates. In comparison to traditional IRT estimation methods, ML2P-VAE achieves competitive performance in terms of accuracy, while providing a speedup in computation time. On high-dimensional datasets, the proposed method yields accurate estimates within reasonable runtime, while traditional methods are unable to produce any result. Stepping away from educational measurement, the architecture of ML2P-VAE provides a high level of interpretability to a neural network model. Additionally, a new VAE architecture is developed which allows for correlated latent code – an idea which can be extended to other fields where prior knowledge of

the abstract latent space is available.

Technology has also had an impact on education which is more visible on the user side – electronic learning environments have been around for a while, but are increasing in popularity and adding more features [44, 72]. Such features include computerized adaptive testing and intelligent tutoring systems, which can tailor content and give feedback to users on a more personal level based on their particular needs [59, 67].

These applications require the ability to evaluate student learning in real-time, updating the estimate of student knowledge $\boldsymbol{\Theta}$ both as more data becomes available and as students learn from their previous experiences. This is the task of knowledge tracing, originally introduced via a Bayesian approach in 1995 [23] to track the progress of students learning to program. More recently, deep neural networks have become the standard in knowledge tracing, including the use of LSTMs [73] and attention-based neural networks [68].

Deep knowledge tracing methods take a partial sequence of student interactions, each of which includes an indicator of the question answered, and whether the item was answered correctly or not. The output is a predicted probability that the student will answer the *next* item in the sequence correctly. While state-of-the-art models are able to approximate this probability with high precision [101], it is the only useful and readily available measure of student knowledge provided by the model, though other post hoc techniques can be applied. This points back towards the previously discussed black-box nature of deep neural networks.

A contribution of this thesis is a modification to existing knowledge tracing methods which links the knowledge tracing framework directly to Item Response Theory. Since the output of all knowledge tracing models is the probability of success on the next interaction, it makes sense to utilize the theory of IRT latent trait models in this application. Using a similar technique as in Part I, the expert-annotated item-skill association is incorporated into the neural architecture, allowing for interpretation of a hidden layer as estimates to a student's latent ability. This explicit representation of $\boldsymbol{\Theta}$ is very convenient in practice, and allows knowledge tracing models to also act as parameter estimation methods. This presents a trade-off between interpretability and accuracy, though the proposed "IRT-inspired knowledge tracing" methods still remain competitive with other models.

1.1 Organization

This thesis is organized into two parts. In Part I, Chapters 2-4 introduce Item Response Theory (IRT) and analyze the novel parameter estimation method, ML2P-VAE. This method uses a modified variational autoencoder to estimation parameters in IRT models. Chapter 2 provides background on two relevant neural network models (autoencoders and variational autoencoders), along with a summary of traditional IRT parameter estimation methods. Chapter 3 describes the ML2P-VAE method in detail, including a software package which was developed for easy implementation of the method. Chapter 4 summarizes all results and experiments performed on educational datasets with ML2P-VAE.

In Part II, Chapters 5-7 explore knowledge tracing, a task commonly seen in online learning environments. While other deep learning methods for knowledge tracing lack interpretability, new methods presented in this thesis sacrifice prediction power for explainability. Chapter 5 gives background on time-dependent neural networks such as recurrent neural networks (RNN), long short-term memory (LSTM) networks, and Transformers, along with a literature review of other popular knowledge tracing methods. Chapter 6 proposes a method of incorporating Item Response Theory models into the knowledge tracing framework. Chapter 7 presents results of the interpretable adaptation to knowledge tracing, including comparisons to other methods in the literature.

In the final chapter of this thesis, Chapter 8 briefly explores some applications outside of education which the ML2P-VAE method can be applied to. These areas include the health sciences, where IRT models have previously been applied to the Beck Depression Inventory [5, 34, 48] and personality questionnaires such as the Big Five [80]. A new extension to sports analytics is explored as well, using ML2P-VAE to evaluate the skill of professional baseball players [20]. Finally, the contributions and significance of this thesis work as a whole is summarized in the conclusion.

CHAPTER 2
NEURAL NETWORKS AND EDUCATIONAL MEASUREMENT

In this chapter we present background information for Part I, variational autoencoders for parameter estimation in Item Response Theory. This includes the architecture of particular neural network models, an overview of educational measurement, and a review of other parameter estimation techniques.

Neural Networks

In recent years, artificial neural networks (ANN) have become an increasingly popular tool for machine learning problems. Though they have been around since the 1960's [83], GPU technology has become more accessible and modern computers are more powerful, allowing any interested researcher to train a basic neural network on a personal machine. ANN can be applied to a diverse set of problems, including regression, classification, computer vision, natural language processing, function approximation, data generation, and more [42] [100]. A brief overview of the mathematical inner workings of ANN is included in Appendix A.

One of the biggest critiques of ANN is their black-box nature, meaning that the decision process of a trained model is typically not explainable by humans. As opposed to simpler methods such as decision trees or linear regression, neural networks are not interpretable. This makes them less desirable in certain applications where researchers wish to know *why* a model outputs a particular prediction in the way that it does. For example, if a financial institution is using data-driven methods to

determine whether or not to approve a loan application, the institution should be able to explain to the customer why they were denied [17]. Further, it is possible that a black-box neural network could learn and use sensitive information such as race, age, or gender in its prediction, which would raise legal questions in the United States [65].

The push for explainable AI has led to multiple approaches to increase model interpretability. Some have aimed to combine deep learning methods with existing interpretable methods, in hopes of increasing the performance of explainable methods without sacrificing its interpretability [38]. Another option is to use a sort of hybrid learning, where interpretable models defer to a black-box model if they are not confident in their prediction [76]. Others have started with deep models and cut back on complexity, making specific modifications which increase interpretability. For example, the loss function of a convolutional neural network can be adapted so that humans can better understand the features extracted in the hidden layers [102].

The field of education is an area which often desires interpretable models. Researchers often need to be able to point out specific details of decisions made by AI. A student deserves an answer to *why* they may have failed a test, and a teacher should be given instructions on *how* to fix the student's misconceptions.

Part I of this thesis introduces the use of ANN models in educational measurement. A modification is described in Chapter 3 which incorporates Item Response Theory and adds interpretability to neural networks. Next, we describe two important types of neural networks – autoencoders and variational autoencoders – which

can be altered to fit the educational measurement application.

2.1 Autoencoders

An autoencoder (AE) is a neural network where the input and output layers are the same shape. The objective for a given data point is to minimize the difference between the output, called the reconstruction, and the input. Typically, the middle hidden layers of an AE are of smaller dimension than the input space. In this way, autoencoders are an unsupervised learning technique for (nonlinear) dimension reduction. Mathematically, we can define an autoencoder in two parts as follows.

For an input $\boldsymbol{x} \in \mathbb{R}^n$, define the *encoder* as a function $f : \mathbb{R}^n \to \mathbb{R}^d$ mapping $\boldsymbol{x} \mapsto \boldsymbol{z} := f(\boldsymbol{x})$. Usually, $d < n$, and $\boldsymbol{z}$ lies in a hidden feature space. The encoder sends an observed data point to its representation in a learned feature space. Define the *decoder* as a function $g : \mathbb{R}^d \to \mathbb{R}^n$ mapping $\boldsymbol{z} \mapsto \hat{\boldsymbol{x}} := g(\boldsymbol{z})$. The decoder maps a hidden representation $\boldsymbol{z}$ to a reconstruction of the original encoder input $\boldsymbol{x}$. Note that in our case, the functions f and g are both parameterized by neural networks, each of which can have any number of hidden layers. The end-to-end autoencoder is then the function composition $\mathcal{A}(\boldsymbol{x}) := g(f(\boldsymbol{x})) : \mathbb{R}^n \to \mathbb{R}^n$, visualized in Figure 2.1.

To train an AE, the loss function minimizes the difference between the input and output (see Appendix A.3). This can be done in a number of ways, including the simple mean squared error loss

$$\mathcal{L}(\boldsymbol{x}) = ||\boldsymbol{x} - g(f(\boldsymbol{x}))||_2^2 \tag{2.1}$$

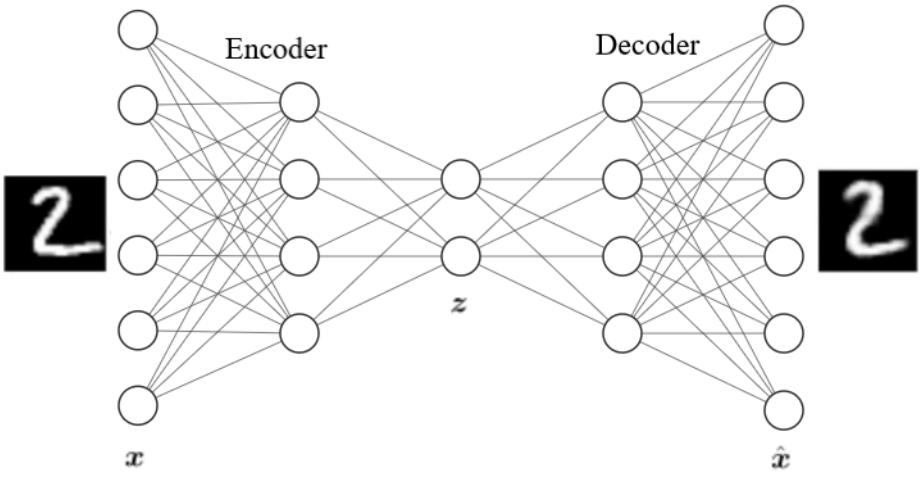

Figure 2.1: Visualization of an autoencoder with $n = 6$ and $d = 2$.

or cross-entropy loss for binary data

$$\mathcal{L}(x) = \sum_{i=1}^{n} -x_i \log(g(f(x_i))) - (1 - x_i) \log(1 - g(f(x_i))). \tag{2.2}$$

Autoencoders with only a single hidden layer can be compared with nonlinear principal components analysis (PCA), and using linear activation functions allows for recovery of PCA loading vectors [74]. AEs have clear applications in image compression straight out-of-the-box, and can be modified for more complicated problems. De-noising autoencoders [95] are capable of processing noisy images and cleaning them up. To do this, the input data is corrupted by deleting pixels at random. Then, the de-noising AE reconstructs the original image from the corrupted data. Autoencoders can also be modified for data generation applications using a variational autoencoder.

2.2 Variational Autoencoders

The motivation for designing a variational autoencoder (VAE), introduced by Kingma and Welling [51], is different from that of a regular autoencoder in that the interest is not rooted specifically to neural networks. Rather, a probabilistic point of view provides the main source of motivation, and ANN are seen as a useful tool used to implement a VAE.

Consider a dataset $\mathbb{X} = \{\boldsymbol{x}_j\}_{j=1}^N$, where each data point $\boldsymbol{x}_j \in \mathbb{R}^n$ is generated by a random process involving an unobserved variable $\boldsymbol{z} \in \mathbb{R}^d$. This unobserved variable is often referred to as "latent code." It is assumed that to generate the observed data point $\boldsymbol{x}_j$, a value $\boldsymbol{z}_j$ is first sampled from a prior distribution $p_z^*(\boldsymbol{z})$, and then $\boldsymbol{x}_j$ is generated from a distribution $p_x^*(\boldsymbol{x}|\boldsymbol{z})$.

From observing the dataset $\mathbb{X}$, the latent variables $\boldsymbol{z}_j$ and the parameters of the true posterior distribution $p_x^*(\boldsymbol{x}|\boldsymbol{z})$ are unknown. The goal of a VAE is to approximate these parameters, which leads to the ability to represent the observed data and generate new data. This should be done with a general algorithm that is unaffected by (i) intractability of the marginal likelihood $p_x^*(\boldsymbol{x}) = \int p_z^*(\boldsymbol{z}) p_x^*(\boldsymbol{x}|\boldsymbol{z}) d\boldsymbol{z}$ and (ii) large amounts of data [51]. Note that (i) is important because if $p_x^*(\boldsymbol{x})$ is intractable and thus $p_z^*(\boldsymbol{z}|\boldsymbol{x})$ is intractable when applying Bayes' Theorem, then the EM algorithm cannot be used – an issue explored further in Section 2.4.2.

In order to implement this task, two neural networks $q_\alpha(\boldsymbol{z}|\boldsymbol{x})$ and $p_\beta(\boldsymbol{x}|\boldsymbol{z})$ (probabilistic encoder and decoder, respectively) are used to approximate the unobservable true posterior distributions $p_z^*(\boldsymbol{z}|\boldsymbol{x})$ and $p_x^*(\boldsymbol{x}|\boldsymbol{z})$. In the encoder/decoder,

the subscripts α and β refer to settings of the trainable parameters of the neural network, while the superscript $*$ indicates the (possibly unknown) true distributions. Note that unlike a regular autoencoder, the probabilistic encoder $q_\alpha(\boldsymbol{z}|\boldsymbol{x})$ of a VAE outputs a posterior probability distribution for $\boldsymbol{z}$ given $\boldsymbol{x}$, rather than a single value.

2.2.1 A Note on Information Theory

Before continuing, we introduce some ideas from information theory: entropy, cross-entropy, and KL-Divergence [9]. Consider a discrete random variable X. For a particular value x, we can compute the amount of information gained from observing x as $h(X) = -\log_2 P(X = x)$. When we use $\log_2$, the unit for information is "bits," but any base for the logarithm can be used. Note that more information is gained from observing a low-probability event than from observing a high-probability event.

Shannon entropy is defined as the expectation of $h(X)$, or the average amount of information that will be learned by observing a random x when X is a discrete random variable [86]. The continuous analogue of Shannon entropy is the limiting density of discrete points [49], but is omitted for simplicity. Here, we consider Shannon's *differential entropy* which has a more intuitive formulation, but is a limiting case which lacks the association with discrete entropy. The differential entropy of a continuous random variable X is given as

$$H[P] = -\int P(x) \log P(x) dx \tag{2.3}$$

Now assume that we also have access to a distribution $Q(X)$ which approximates a possibly unknown $P(X)$. *Cross-entropy* is the average amount of information

needed to identify an event x which was drawn from the approximate distribution $Q(X)$ instead of drawn from the true distribution $P(X)$. Cross-entropy is given as

$$H[P, Q] = -\int P(x) \log Q(x) dx \tag{2.4}$$

Finally, define *Kullback-Leibler Divergence* (KL-Divergence) [53] as

$$\mathcal{D}_{KL}\left[P\middle||Q\right] = H[P] - H[P, Q] = -\int P(x) \log\left(\frac{Q(x)}{P(x)}\right) dx \tag{2.5}$$

Intuitively, KL-Divergence is the average amount of information that is lost if the approximating distribution $Q(X)$ is used instead of the true distribution $P(X)$. KL-Divergence cannot be interpreted as a metric or as a distance between the distributions $P(X)$ and $Q(X)$ because it is not symmetric – in general, $\mathcal{D}_{KL}[P||Q] \neq \mathcal{D}_{KL}[Q||P]$. KL-Divergence is non-negative [51], and $\mathcal{D}_{KL}[P||Q] = 0$ if and only if $P(X) = Q(X)$ almost everywhere.

2.2.2 VAE Derivation

We derive the desired loss function for a VAE. The log marginal likelihood of N observed data points is given as

$$\log p_x^*(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) = \sum_{j=1}^{N} \log p_x^*(\boldsymbol{x}_j) \tag{2.6}$$

Denoting $\boldsymbol{x} = \boldsymbol{x}_j$, we can rewrite each $\log p_x^*(\boldsymbol{x}_j)$ term as

$$\begin{aligned}
\log p_x^*(\boldsymbol{x}) &= \int q_\alpha(\boldsymbol{z}|\boldsymbol{x}) \log p_x^*(\boldsymbol{x}) d\boldsymbol{z} \\
&= \int q_\alpha(\boldsymbol{z}|\boldsymbol{x}) \log \left(\frac{p_z^*(\boldsymbol{z}|\boldsymbol{x}) p_x^*(\boldsymbol{x})}{p_z^*(\boldsymbol{z}|\boldsymbol{x})} \right) d\boldsymbol{z} \\
&= \int q_\alpha(\boldsymbol{z}|\boldsymbol{x}) \log \left(\frac{p^*(\boldsymbol{x}, \boldsymbol{z})}{p_z^*(\boldsymbol{z}|\boldsymbol{x})} \right) d\boldsymbol{z} \\
&= \int q_\alpha(\boldsymbol{z}|\boldsymbol{x}) \left(\log \frac{q_\alpha(\boldsymbol{z}|\boldsymbol{x})}{p_z^*(\boldsymbol{z}|\boldsymbol{x})} + \log \frac{p^*(\boldsymbol{x}, \boldsymbol{z})}{q_\alpha(\boldsymbol{z}|\boldsymbol{x})} \right) d\boldsymbol{z} \\
&= \mathcal{D}_{KL} \left[q_\alpha(\cdot|\boldsymbol{x}) \middle|\middle| p_z^*(\cdot|\boldsymbol{x}) \right] + \int q_\alpha(\boldsymbol{z}|\boldsymbol{x}) \log \left(\frac{p^*(\boldsymbol{x}, \boldsymbol{z})}{q_\alpha(\boldsymbol{z}|\boldsymbol{x})} \right) d\boldsymbol{z} \\
&= \mathcal{D}_{KL} \left[q_\alpha(\cdot|\boldsymbol{x}) \middle|\middle| p_z^*(\cdot|\boldsymbol{x}) \right] + \mathbb{E}_{\boldsymbol{z} \sim q_\alpha(\cdot|\boldsymbol{x})} \left[-\log q_\alpha(\boldsymbol{z}|\boldsymbol{x}) + \log p^*(\boldsymbol{x}, \boldsymbol{z}) \right] \\
&= \mathcal{D}_{KL} \left[q_\alpha(\cdot|\boldsymbol{x}) \middle|\middle| p_z^*(\cdot|\boldsymbol{x}) \right] + \tilde{\mathcal{L}}_*(\alpha; \boldsymbol{x})
\end{aligned} \tag{2.7}$$

Note that in the final line, the first term is the KL-Divergence between the approximate and true posterior. Since we don't know the true posterior, we can't calculate this term. But notice that since KL-Divergence is always positive (i.e. $\forall x \in \mathbb{R}$, $\mathcal{D}_{KL}[P||Q] + x \geq x$), it can be ignored and arrive at

$$\begin{aligned}
\log p_x^*(\boldsymbol{x}) \geq \tilde{\mathcal{L}}_*(\alpha; \boldsymbol{x}) &= \mathbb{E}_{\boldsymbol{z} \sim q_\alpha(\cdot|\boldsymbol{x})} \left[-\log q_\alpha(\boldsymbol{z}|\boldsymbol{x}) + \log p^*(\boldsymbol{x}, \boldsymbol{z}) \right] \\
&= \mathbb{E}_{\boldsymbol{z} \sim q_\alpha(\cdot|\boldsymbol{x})} \left[-\log q_\alpha(\boldsymbol{z}|\boldsymbol{x}) + \log p_x^*(\boldsymbol{x}|\boldsymbol{z}) + \log p_z^*(\boldsymbol{z}) \right] \\
&\approx \mathbb{E}_{\boldsymbol{z} \sim q_\alpha(\cdot|\boldsymbol{x})} \left[-\log q_\alpha(\boldsymbol{z}|\boldsymbol{x}) + \log p_\beta(\boldsymbol{x}|\boldsymbol{z}) + \log p_z^*(\boldsymbol{z}) \right] \\
&= -\mathcal{D}_{KL} \left[q_\alpha(\cdot|\boldsymbol{x}) \middle|\middle| p_z^*(\cdot) \right] + \mathbb{E}_{\boldsymbol{z} \sim q_\alpha(\cdot|\boldsymbol{x})} \left[\log p_\beta(\boldsymbol{x}|\boldsymbol{z}) \right] \\
&= \tilde{\boldsymbol{\mathcal{L}}}(\alpha, \beta; \boldsymbol{x})
\end{aligned} \tag{2.8}$$

where the true posterior distribution $p_x^*(\boldsymbol{x}|\boldsymbol{z})$ is approximated by the VAE decoder $p_\beta(\boldsymbol{x}|\boldsymbol{z})$ in the third line of Equation 2.8. Note that all unknown distributions have been replaced by the encoder, decoder, and assumed prior $p_z^*(\boldsymbol{z})$.

The term $\tilde{\mathcal{L}}(\alpha, \beta; \boldsymbol{x})$ is referred to as the variational lower bound or Evidence Lower Bound (ELBO). Increasing the ELBO by varying the parameters α and β will increase the marginal likelihood $\log p_x^*(\boldsymbol{x})$, even though we ignore the term $\mathcal{D}_{KL}\left[q_\alpha(\cdot|\boldsymbol{x})\middle\|p_z^*(\cdot|\boldsymbol{x})\right]$ in Equation 2.8 [103].

We take the ELBO $\tilde{\mathcal{L}}(\alpha, \beta; \boldsymbol{x})$ to be a potential VAE objective function which we wish to maximize. In Equation 2.8, the first term gives the negative KL-Divergence between the probabilistic encoder $q_\alpha(\boldsymbol{z}|\boldsymbol{x})$ and the true prior distribution of the latent code $p_z^*(\boldsymbol{z})$. Note that unlike the true posterior $p_z^*(\boldsymbol{z}|\boldsymbol{x})$, the true prior $p_z^*(\boldsymbol{z})$ is assumed to be known, and it is nearly always assumed to be independent Gaussian.

For now, we follow suit and assume that $\boldsymbol{z} \sim p_z^*(\cdot) = \mathcal{N}(0, I)$, so we structure the encoder to output a standard normal distribution. In Section 3.1.1, we propose a novel neural network architecture which generalizes to $\boldsymbol{z} \sim \mathcal{N}(\mu, \Sigma)$ and allows for the encoder to output a multivariate Gaussian distribution.

These assumptions make computing $\tilde{\mathcal{L}}(\alpha, \beta; \boldsymbol{x})$ much easier. It can be shown [30] that the KL-Divergence between an independent Gaussian distribution and a standard normal distribution of dimension K is calculated as

$$\mathcal{D}_{KL}\left[\mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\sigma}_0^2 I)\middle\|\mathcal{N}(0, I)\right] = \frac{1}{2}\sum_{k=1}^{K}\left(\mu_k^2 + \sigma_{0,k}^2 - 1 - \log(\sigma_{0,k}^2)\right) \tag{2.9}$$

where the vector $\boldsymbol{\sigma}_0^2$ holds the variances $\sigma_{0,k}^2$ (i.e. the squared standard deviation of each dimension). Note that the vectors $\boldsymbol{\mu}_0$ and $\boldsymbol{\sigma}_0^2$ are outputted by the encoder $q_\alpha(\boldsymbol{z}|\boldsymbol{x}_0)$, given the observed input $\boldsymbol{x}_0$. Since Equation 2.9 is in closed form, there is no difficulty in calculating the (possibly high-dimensional) integral that is usually required to compute KL-Divergence.

The second term in Equation 2.8 is similar to Equation 2.4, and depends on the probabilistic decoder $p_\beta(\boldsymbol{x}|\boldsymbol{z})$, which is usually assumed to be either Gaussian or Bernoulli. Considering the desired application of educational measurement where data is given as binary responses, we assume that the decoder outputs a Bernoulli distribution. To estimate the expectation over q_α, we simply sample L times from $q_\alpha(\boldsymbol{z}|\boldsymbol{x})$. In practice, it is often simplest to just choose $L = 1$ [51]. So then

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{z}\sim q_\alpha(\cdot|\boldsymbol{x})}\left[\log p_\beta(\boldsymbol{x}|\boldsymbol{z})\right] &\approx \frac{1}{L}\sum_{l=1}^{L}\log p_\beta(\boldsymbol{x}|\boldsymbol{z}^{(l)}) \\
&\approx \log p_\beta(\boldsymbol{x}|\boldsymbol{z}^*) \\
&= \log\left(\prod_{i=1}^{n} p_\beta(x_i = 1|\boldsymbol{z}^*)^{x_i}\cdot p_\beta(x_i = 0|\boldsymbol{z}^*)^{x_i}\right) \\
&= \sum_{i=1}^{n} x_i \log\hat{x}_i + (1 - x_i)\log(1 - \hat{x}_i)
\end{aligned} \tag{2.10}$$

where $\hat{x}_i = p_\beta(x_i = 1|\boldsymbol{z}^*)$ gives $\hat{\boldsymbol{x}} = \{\hat{x}_i\}_{i=1}^n$, the reconstruction of the input $\boldsymbol{x}$ from the result of the encoder q_α. Note that the final line of Equation 2.10 results in the negative binary cross-entropy loss function commonly used in classification problems.

The process is summarized as follows: given an input vector $\boldsymbol{x}_0 \in \mathbb{R}^n$, we obtain the posterior distribution $q_\alpha(\boldsymbol{z}|\boldsymbol{x}_0)$ (this is done by feeding $\boldsymbol{x}_0$ through the encoder neural network). Sample $\boldsymbol{z}^* \sim q_\alpha(\cdot|\boldsymbol{x}_0)$, and compute $\hat{\boldsymbol{x}} = p_\beta(\boldsymbol{x}|\boldsymbol{z}^*)$ (this is done by feeding $\boldsymbol{z}^*$ through the decoder neural network). As demonstrated in Equation 2.9 and Equation 2.10, we can easily compute each term of the objective function $\tilde{\boldsymbol{\mathcal{L}}}(\alpha, \beta; \boldsymbol{x}_0)$.

Usually when working with neural networks, the goal is to minimize a loss function, rather than maximize an objective function. As such, we define the loss

function

$$\begin{aligned}\mathcal{L}(\alpha, \beta; \boldsymbol{x}) &= -\tilde{\mathcal{L}}(\alpha, \beta; \boldsymbol{x}) \\ &= \Big(\sum_{i=1}^{n} -x_i \log p_\beta(x_i|\boldsymbol{z}^*) - (1 - x_i) \log(1 - p_\beta(x_i|\boldsymbol{z}^*)) \Big) + \mathcal{D}_{KL} \left[q_\alpha(\cdot|\boldsymbol{x}) \big|\big| p_z^*(\cdot) \right]\end{aligned} \tag{2.11}$$

where $\boldsymbol{z}^* \sim q_\alpha(\cdot|\boldsymbol{x})$. The parameters α and β represent the weights and biases of the encoder and decoder, and are updated via a gradient descent algorithm in order to minimize $\mathcal{L}(\alpha, \beta; \boldsymbol{x})$ given in Equation 2.11. Note that $\mathcal{L}(\alpha, \beta; \boldsymbol{x})$ can simply be understood as reconstructing a binary input $\boldsymbol{x}$ via the first term (which mirrors the binary cross-entropy loss function in Equation 2.2), while regularizing on the latent code $\boldsymbol{z}$ via the second term.

2.2.3 Implementation Details

Recall that each observed data point $\boldsymbol{x}_j \in \mathbb{R}^n$ and the corresponding latent code $\boldsymbol{z}_j \in \mathbb{R}^d$. To parameterize the encoder $q_\alpha(\boldsymbol{z}|\boldsymbol{x})$ as a neural network, an input layer with n nodes is required. The encoder can (but does not need to) include a number of hidden layers of varying size. The output of the encoder must include $2 \cdot d$ nodes, assuming that $p_z^*(\boldsymbol{z}) = \mathcal{N}(0, I)$.

Notice that in Equation 2.9, there is a term $\log(\sigma_{0,k}^2)$, dependent on input $\boldsymbol{x}_0$ and all parameters of the encoder. If $\sigma_{0,k}^2 \leq 0$ at any point during training for any k, then the VAE loss would not be possible to calculate. To avoid this issue and ensure that $\sigma_{0,k}^2 > 0$ regardless of inputs or encoder parameters, the encoder outputs the log-variances rather than the variances. The first d nodes outputted by

the encoder represent the latent mean vector $\boldsymbol{\mu}$, and the last d nodes represent the log latent variances $\log \boldsymbol{\sigma}^2$. So for an observed input $\boldsymbol{x}_0$, the encoder will output a d-dimensional standard normal distribution $\mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\sigma}_0^2 I)$.

The input layer of the decoder $p_\beta(\boldsymbol{x}|\boldsymbol{z})$ has d nodes and takes in a sample from $\mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\sigma}_0^2 I)$, given the original input $\boldsymbol{x}_0$. But the sampling operation is not deterministic, posing a problem for backpropagation-based gradient descent. A re-parameterization is used: sample $\boldsymbol{\varepsilon}_0 \sim \mathcal{N}(0, I)$, then calculate $\boldsymbol{z}_0 = \boldsymbol{\mu}_0 + \boldsymbol{\varepsilon}_0 \odot \boldsymbol{\sigma}_0^2$ where $\odot$ is element-wise vector multiplication. Then $\boldsymbol{z}_0$ is fed through a number of hidden layers to an output layer of size n. The output can be interpreted as the reconstruction $\hat{\boldsymbol{x}}_0$.

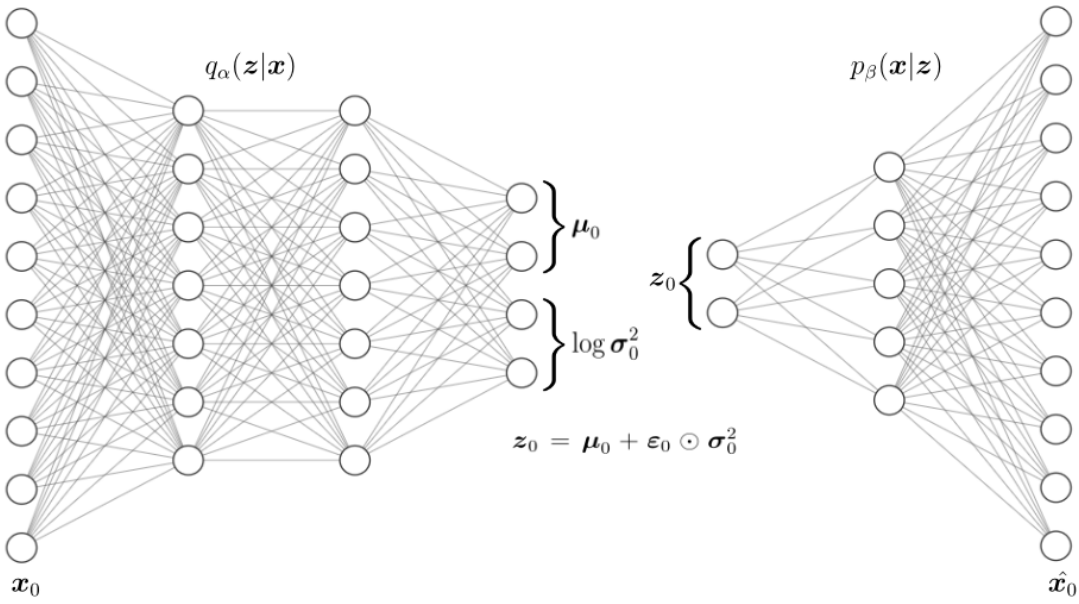

Figure 2.2: Visualization of a VAE architecture with $n = 10$ and $d = 2$.

The VAE architecture is summarized in Figure 2.2. Note that the VAE does not need to be symmetric – the encoder and decoder can have a different number of hidden layers of different sizes.

Educational Measurement

In educational measurement, a common goal is to quantify the knowledge of students from the results of some assessment. In a classroom setting, grades are typically assigned based on the percentage of questions answered correctly by a student on assignments. The letter grades assigned from these percentages can serve as a naive measure of student knowledge – "A" students have completely mastered the material, "B" students have a good grasp of material, "C" students are fairly average, and "D" and "F" students have significant gaps in their knowledge.

The practice of evaluating student ability purely from a raw percentage score is known as classical test theory (CTT) [93]. But there are clear issues with this approach. Not all questions on an exam or homework assignment are created equally – some questions are easier, and some more difficult. Consider a scenario where two students both answer 17 out of 20 questions correctly on a test for a raw score of 85%. But if Student A answered questions 1, 8, and 9 wrongly while Student B answered 4, 17, and 20 incorrectly, it is not likely that that Student A and Student B possess the same level of knowledge. For example, questions 1, 8, and 9 could be much more difficult than questions 4, 17, and 20, or Student B may have guessed correctly on some items. Additionally, the two sets of problems could cover different types of

material. CTT does not account for either of these situations, and naively quantifies the knowledge of Student A and Student B as equal.

More sophisticated methods have been developed which attempt to more accurately quantify student learning. Cognitive Diagnostic Models (CDM) aim to classify whether students possess mastery of a given skill or not [89]. This discrete classification can be useful in determining whether or not a student meets a prerequisite, or in deciding if the student is proficient enough to move on to the next level of coursework. We focus instead on Item Response Theory, where student knowledge is assumed to be continuous. In this case, a student's latent ability is quantified as a real number lying within some interval, such as $(-3, 3)$. A value near the middle of the interval translates to a student with average knowledge, and a large (resp. small) value corresponds with a high (resp. low) level of understanding.

2.3 Item Response Theory

Item Response Theory (IRT) is a field of quantitative psychology which uses statistical models to model student knowledge [57]. These models often give the probability of a question being answered correctly as a function of the student's ability. In IRT, it is assumed that each student, indexed by j, possesses some continuous latent ability θ_j. The term "latent ability" is synonymous with "knowledge," "trait," or "skill." Often, it is assumed that amongst the population of students, $\theta_j \sim \mathcal{N}(0, 1)$ [93].

In this work, we often consider the multidimensional case where each student

has multiple latent abilities. For example, in the context of an elementary math exam, we may wish to measure the four distinct skills "add", "subtract", "multiply", and "divide." This scenario is referred to as multidimensional item response theory, and we write the set of student j's K latent abilities as a vector $\boldsymbol{\Theta}_j = (\theta_{1j}, \theta_{2j}, \ldots, \theta_{Kj})^\top$. It is then assumed that the latent abilities of students follow some multivariate Gaussian distribution, $\mathcal{N}(0, \Sigma)$. For simplicity, the covariance matrix Σ is often taken to be the identity matrix, making each latent skill independent of one another. This assumption on Σ gives practical advantages in estimation, but is often not realistic in real-world applications.

Note that $\boldsymbol{\Theta}_j$ is not directly observable in any way. Given a student's responses to an assessment with n items, a common task is to infer their knowledge $\boldsymbol{\Theta}_j$. Student j's set of responses can be written as a binary n-dimensional vector $\boldsymbol{u}_j = (u_{1j}, u_{2j}, \ldots, u_{nj})^\top$, where

$$u_{ij} = \begin{cases} 1 & \text{if student } j \text{ answers item } i \text{ correctly} \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

IRT models aim to model the probability of a student answering a particular question correctly, so that the probability of student j answering item i correctly is given by some function of $\boldsymbol{\Theta}_j$:

$$P(u_{ij} = 1 | \boldsymbol{\Theta}_j) = f(\boldsymbol{\Theta}_j; V_i) \tag{2.13}$$

where V_i is a set of parameters associated with item i. In general, $f : \mathbb{R}^K \to [0, 1]$, where K is the number of latent abilities, a continuous function which is strictly increasing with respect to $\boldsymbol{\Theta}_j$. Often, the function f follows a curve similar to what

is shown in Figure 2.3.

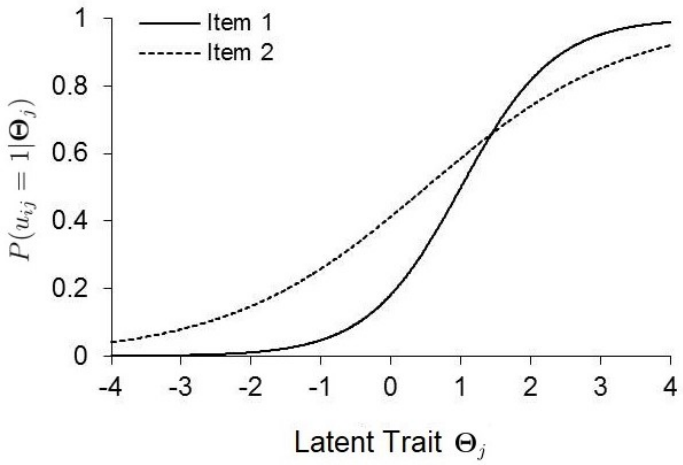

Figure 2.3: An item characteristic curve visualizes the relation between a student's ability and the probability of answering an item correctly.

It is worth mentioning that IRT models also exist for non-binary response data. For example, Samejima's graded response model [85] allows for items to be answered in multiple categories or grades. This idea is related to partial credit scoring [58], where students can receive a fraction of the points available on a single question if they demonstrate some knowledge of how to answer the question correctly. In this case, u_{ij} could hold the possible values $\{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\}$, and Equation 2.13 could be

characterized as

$$\begin{aligned} P(u_{ij} \geq 0|\mathbf{\Theta}_j) &= 1 \\ P(u_{ij} \geq \frac{1}{4}|\mathbf{\Theta}_j) &= f_1(\mathbf{\Theta}_j; V_i) \\ P(u_{ij} \geq \frac{1}{2}|\mathbf{\Theta}_j) &= f_2(\mathbf{\Theta}_j; V_i) \\ P(u_{ij} \geq \frac{3}{4}|\mathbf{\Theta}_j) &= f_3(\mathbf{\Theta}_j; V_i) \\ P(u_{ij} = 1|\mathbf{\Theta}_j) &= f_4(\mathbf{\Theta}_j; V_i) \end{aligned} \tag{2.14}$$

In the following sections, we describe various candidates for the function f. The focus is on dichotomous responses as in 2.12, but can be extended to the graded response model. Though each is presented in the context of single-dimensional IRT ($K = 1$), they can all be easily adapted to higher dimensions where $\mathbf{\Theta}_j$ is a vector.

2.3.1 Rasch Model

One of the first IRT models was proposed by Georg Rasch in 1960 [77]. Rasch asserted that the probability of a student answering an item correctly is a function of the ratio ξ/δ, where $\xi > 0$ represents the student's knowledge, and $\delta > 0$ quantifies the difficulty of an item. Consider the value $\dfrac{\xi}{\xi + \delta} = \dfrac{1}{1 + \delta/\xi}$ and note that $\dfrac{\xi}{\xi + \delta} \to 1$ as $\xi \to \infty$. After the re-parameterization $\xi = e^{\theta}$ and $\delta = e^{b}$, we arrive at the 1-Parameter Logistic Model, often referred to as the Rasch Model [93].

$$\begin{aligned} P(u_{ij} = 1|\xi_j; \delta_i) &= \frac{1}{1 + \delta_i/\xi_j} = \frac{1}{1 + e^{b_i}/e^{\theta_j}} \\ P(u_{ij} = 1|\theta_j; b_i) &= \frac{1}{1 + e^{b_i - \theta_j}} \end{aligned} \tag{2.15}$$

Note that $\theta \in \mathbb{R}$ and $b \in \mathbb{R}$ still represent student ability and item difficulty, respectively. We can interpret the difficulty parameter b as a threshold: when $\theta = b$,

then the student has a 50% chance of answering the question correctly. The plot shown in Figure 2.3, the item characteristic curve (ICC) of two items are shown. The role of the difficulty parameter b_i is to transpose the curve to the left (for an easier item) or to the right (for a more difficult item). The slope of the ICC is unaffected by the difficulty parameter.

2.3.2 Normal Ogive Model

A slightly more sophisticated method for measuring student performance is the normal ogive model. We introduce a discrimination parameter, a_i, which quantifies the capability of item i in distinguishing between students who have or have not mastered the knowledge concept θ [93]. In other words, a_i tells *how much* of skill θ is required to answer item i correctly. The effect of different quantities of a_i is seen in Figure 2.3 – the curve for Item 1 has a steeper slope than that of Item 2, due to Item 1 having a larger discrimination parameter than that of Item 2.

The normal ogive model gives the probability of student j answering item i correctly as

$$P(u_{ij} = 1|\theta_j; a_i, b_i) = \frac{1}{\sqrt{2\pi}} \int_{-a_i\theta_j+b_i}^{\infty} e^{\frac{-z^2}{2}} dz \tag{2.16}$$

Note the similarity between Equation 2.16 and the cumulative distribution function for a Gaussian distribution. The normal ogive model is popular among statisticians for this reason, but can be difficult to use for parameter estimation due to its complicated form.

2.3.3 2-Parameter Logistic Model

The model which this work focuses on most is the 2-parameter logistic (2PL) model. Like the normal ogive model, the 2PL model uses both the discrimination and difficulty item parameters. The probability of student j answering item i correctly is given by

$$P(u_{ij} = 1|\theta_j; a_i, b_i) = \frac{1}{1 + e^{-a_i\theta_j + b_i}} \tag{2.17}$$

Equation 2.17 has the same form (a logistic curve) as that of the Rasch model in Equation 2.15, but adds in the discrimination parameter a_i which serves the same purpose as the discrimination parameter of the normal ogive model in Section 2.3.2. If this parameter is scaled by 1.7, then the ICC from the normal ogive model differs from that of the 2PL model by 0.01 uniformly [4]. In a sense, we can consider the 2PL model to be a very good approximation of the normal ogive model. Due to the simple form of Equation 2.17, using this model makes parameter estimation much easier than the normal ogive model.

2.3.4 Multidimensional Item Response Theory

The previously described statistical models can all be extended so that each student possesses K latent traits, instead of a single quantity θ. In multidimensional item response theory (MIRT), models give the probability of a correct answer as a function of student j's ability vector $\mathbf{\Theta}_j = (\theta_{j1}, \ldots, \theta_{jK})^\top$. The generalization of 2.17

is given by the multidimensional logistic 2-parameter (ML2P) model:

$$P(u_{ij} = 1|\boldsymbol{\Theta}_j; \boldsymbol{a_i}, b_i) = \frac{1}{1 + \exp\left(-\boldsymbol{a_i}^\top \boldsymbol{\Theta}_j + b_i\right)} = \frac{1}{1 + \exp\left(-\sum_{k=1}^{K} a_{ik}\theta_{jk} + b_i\right)} \tag{2.18}$$

Here, the discrimination parameters $\boldsymbol{a_i} \in \mathbb{R}^K$ are given as vector, where each entry $a_{ik} \in \boldsymbol{a_i}$ quantifies *how much* of skill k is required to answer item i correctly. The difficulty parameter remains as a single value b_i for each item – difficulty is not partitioned to be skill-specific. The ML2P model is the main focus of this thesis.

A few summary statistics for items modeled by MIRT are often of interest [78]. The multidimensional discrimination power of item i is given as

$$MDISC_i = \sqrt{\sum_{k=1}^{K} a_{ik}^2} \tag{2.19}$$

and the multidimensional difficulty of item i is

$$MDIFF_i = \frac{b_i}{MDISC_i}. \tag{2.20}$$

Equations 2.19 and 2.20 give a more direct connection with the a and b parameters in the unidimensional case described in Equation 2.18. Large positive (resp. negative) values of $MDIFF_i$ indicate difficult (resp. easy) items.

In MIRT, it is convenient to notate the relationship between skills and items with a binary matrix. Define the Q-matrix [91] $Q \in \{0, 1\}^{n \times K}$ so that

$$q_{ik} = \begin{cases} 1 & \text{if item } i \text{ requires skill } k \\ 0 & \text{otherwise} \end{cases} \tag{2.21}$$

In real applications, the Q-matrix is annotated by an expert in the field, as it is usually possible to discern the concepts needed to answer an item correctly. In

relation to the ML2P model (Equation 2.18), notice that if $q_{ik} = 0$, then $a_{ik} = 0$ as well. A convenient formulation of the ML2P model was given by Silva et al. [25]:

$$P(u_{ij} = 1|\boldsymbol{\Theta}_j; \boldsymbol{a_i}, b_i) = \frac{1}{1 + \exp\left(-\sum_{k=1}^{K} q_{ik} a_{ik} \theta_{jk} + b_i\right)} \tag{2.22}$$

Though experts can produce a Q-matrix for a given assessment, the more precise matrix of discrimination parameters $A = [a_{ik}]$ can not be discovered so easily.

2.4 IRT Parameter Estimation

IRT models provide a way of predicting student performance on an assessment, given their latent abilities. However, this is not always of immediate use in practice – in the real world, continuous representations of student ability remain hidden, and the value of item parameters (e.g. difficulty) are not readily accessible.

Often, IRT models are used in a post-assessment setting, rather than as a prediction tool. Given the student's responses (often a binary vector indicating correct/incorrect answers), the goal is to infer their continuous latent abilities $\boldsymbol{\Theta}$ of students and the item parameters. Simpler cases can be considered; if an established assessment with known item parameters is given to a new population of students, then only the student parameters need to be estimated. The accepted item parameter values can be used in this estimation to provide a more accurate measure of student's abilities. Likewise, a new assessment could be given to a group of students whose latent skills are already known. That student-specific information can be used to infer estimates to item parameters.

In this section, we focus on methods which assume zero knowledge of student

or item parameters. As such, all parameters must be estimated jointly, which has shown to be a difficult task when a large number of items, students, and latent traits are present [13].

Since the only available information is the set of binary responses for a number of students, the function to optimize is the log-likelihood of the observed data. Let $U \in \mathbb{R}^{n \times N}$ be the binary matrix of N student responses to an assessment with n items, where $u_{ij} = 1$ if student j answered item i correctly, and 0 otherwise as described in Equation 2.12. Denote $P_{ij} = P(u_{ij} = 1 | \boldsymbol{\Theta}_j \boldsymbol{a}_i, b_i)$ as the probability of student j answering item i correctly – we focus on the ML2P model in Equation 2.18.

Then we have the likelihood function [4] for a all student responses as

$$L = \prod_{j=1}^{N} \prod_{i=1}^{n} P_{ij}^{u_{ij}} \cdot (1 - P_{ij})^{1 - u_{ij}} \tag{2.23}$$

As is common in other machine learning applications we do not directly maximize L, but rather the log-likelihood function

$$\log L = \sum_{j=1}^{N} \sum_{i=1}^{n} u_{ij} \log P_{ij} + (1 - u_{ij}) \log(1 - P_{ij}) \tag{2.24}$$

Notice the similarity of Equation 2.24 to the VAE loss function described in Equation 2.11. In the following sections, we detail other methods for IRT parameter estimation and their shortcomings in dealing with high-dimensional data.

2.4.1 Joint Maximum Likelihood Estimation

Maximum Likelihood Estimation (MLE) is a popular method for parameter estimation in many fields, not just psychology or education. In this application, we

refer to this method as Joint Maximum Likelihood Estimation (JMLE), since both the item and student parameters are unknown. The high-level goal of parameter estimation is to find the parameters of a probability distribution which are most likely to produce the observed samples [63]. These parameters are found by directly computing the gradient of Equation 2.24 with respect to every individual parameter θ_{jk}, a_{ik}, and b_i, then solving a system of equations. For example, we must compute

$$\frac{\partial \log L}{\partial \theta_{jk}}, \quad \frac{\partial \log L}{\partial a_{ik}}, \quad \frac{\partial \log L}{\partial b_i}$$

for $1 \leq i \leq n$, $1 \leq j \leq N$, and $1 \leq k \leq K$. Setting each partial derivative equal to zero yields a system of $NK + nK + n$ equations with $NK + nK + n$ unknowns:

$$\left[\frac{\partial \log L}{\partial \theta_{11}}, \frac{\partial \log L}{\partial \theta_{12}}, \ldots, \frac{\partial \log L}{\partial \theta_{NK}}, \frac{\partial \log L}{\partial a_{11}}, \frac{\partial \log L}{\partial a_{12}}, \ldots, \frac{\partial \log L}{\partial a_{nK}}, \frac{\partial \log L}{\partial b_1}, \ldots, \frac{\partial \log L}{\partial b_n}\right]^\top = \mathbf{0} \tag{2.25}$$

Due to the size of this system of equations (which can scale in three different dimensions N, K, and n), the system is solved iteratively via Newton-Raphson methods [6]. Denote the vector of parameter estimates as $\boldsymbol{x} \in \mathbb{R}^{NK+nK+n}$, the gradient vector in Equation 2.25 as $\boldsymbol{f}$, and the matrix $J \in \mathbb{R}^{(NK+nK+n)\times(NK+nK+n)}$ to be the Jacobian of $\boldsymbol{f}$, which holds all second-order partial derivatives of $\log L$. Note that J and $\boldsymbol{f}$ are functions which require an input $\boldsymbol{x}$.

Given an initial guess $\boldsymbol{x}_0$, the Newton-Raphson iteration is defined as

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - J(\boldsymbol{x}_t)^{-1}\boldsymbol{f}(\boldsymbol{x}_t) \tag{2.26}$$

This is where we first encounter a difficulty with dimensionality. The matrix J is quite large – it has dimension $(NK + nK + n) \times (NK + nK + n)$. Inverting this

large matrix, as required in Equation 2.26 quickly becomes difficult – especially when $J(\boldsymbol{x}_t)^{-1}$ needs to be calculated for each iteration.

The structure of the Jacobian J can be organized as

$$J = \begin{bmatrix} A & C \\ C^\top & B \end{bmatrix} \tag{2.27}$$

where $A \in \mathbb{R}^{NK \times NK}$ holds cross-derivatives between N students with K latent abilities, $B \in \mathbb{R}^{(nK+n)\times(nK+n)}$ holds cross-derivatives between n items measuring K latent abilities, and $C \in \mathbb{R}^{NK\times(nK+n)}$ holds the cross-derivatives between students and items.

As such, a number of simplifications must be made. Notice that students are independent of one another (the ability of student j does not affect the ability of student j'), so each cross-derivative between different students is zero. In other words, $\dfrac{\partial^2 \log L}{\partial\theta_{jk}\partial\theta_{j'l}} = 0$ for all $j \neq j'$. This makes A have block-diagonal form, where each of the N sub-blocks is of size $K \times K$.

In a similar manner, note that items are independent of one another and cross-derivatives between two variables which relate to different items will be zero. The difficulty/discrimination of item i does not affect the difficulty/discrimination of item i': $\dfrac{\partial^2 \log L}{\partial b_i \partial b_{i'}} = 0$ for $i \neq i'$ (the same applies to discrimination parameters a_{ik}). Now B has block-diagonal form, with each of the n sub-blocks has dimension $(K+1)\times(K+1)$.

These two simplifications are straightforward, as we can clearly see in Equation 2.24 that summing over j and i will cause each cross derivative between $j \neq j'$ and $i \neq i'$ to be zero. But the Jacobian J is still not sparse enough – the bottom left and top right blocks $C^\top$ and C are fully populated. To simplify the computational problem this presents, it is assumed that the cross-derivatives between students and

items are fixed to be zero: $\dfrac{\partial^2 \log L}{\partial \theta_{jk} \partial b_i} := 0$ for all j, k, i.

The logic behind this assumption is that "there is no reason to believe that there should be any covariation between an individual examinee and either of the parameters of a given item" [4]. However, it can be seen from Equation 2.24 that this is not necessarily true. We can easily show for the ML2P model in Equation 2.18 that

$$\begin{aligned} \frac{\partial P_{ij}}{\partial a_{ik}} &= \theta_{jk} P_{ij}(1 - P_{ij}) \\ \frac{\partial P_{ij}}{\partial \theta_{jk}} &= a_{ik} P_{ij}(1 - P_{ij}) \\ \frac{\partial^2 P_{ij}}{\partial a_{ik} \partial \theta_{jk}} &= P_{ij}(1 - P_{ij}) \left[1 + a_{ik}\theta_{jk}(1 - 2P_{ij})\right] \end{aligned} \tag{2.28}$$

Then we can calculate

$$\begin{aligned} \frac{\partial \log L}{\partial \theta_{jk}} &= \sum_{i=1}^{n} \frac{u_{ij}}{P_{ij}} \cdot \frac{\partial P_{ij}}{\partial \theta_{jk}} + \frac{1 - u_{ij}}{1 - P_{ij}} \cdot \frac{-\partial P_{ij}}{\partial \theta_{jk}} \quad \text{and} \\ \frac{\partial^2 \log L}{\partial \theta_{jk} \partial a_{ik}} &= \frac{-u_{ij}}{P_{ij}^2} \cdot \frac{\partial P_{ij}}{\partial a_{ik}} \cdot \frac{\partial P_{ij}}{\partial \theta_{jk}} + \frac{u_{ij}}{P_{ij}} \cdot \frac{\partial^2 P_{ij}}{\partial \theta_{jk} \partial a_{ik}} \\ &\quad + \frac{1 - u_{ij}}{(1 - P_{ij})^2} \cdot \frac{\partial P_{ij}}{\partial a_{ik}} \cdot \frac{-\partial P_{ij}}{\partial \theta_{jk}} + \frac{1 - u_{ij}}{1 - P_{ij}} \cdot \frac{-\partial^2 P_{ij}}{\partial \theta_{jk} \partial a_{ik}} \end{aligned} \tag{2.29}$$

$$\Longrightarrow$$

$$\begin{aligned} \frac{\partial^2 \log L}{\partial \theta_{jk} \partial a_{ik}} &= -u_{ij} a_{ik} \theta_{jk} (1 - P_{ij})^2 + u_{ij}(1 - P_{ij}) \left[1 + a_{ik}\theta_{jk}(1 - 2P_{ij})\right] \\ &\quad - (1 - u_{ij}) a_{ik} \theta_{jk} P_{ij}^2 - (1 - u_{ij}) P_{ij} \left[1 + a_{ik}\theta_{jk}(1 - 2P_{ij})\right] \end{aligned}$$

Note that $u_{ij} \in \{0, 1\}$, giving two simpler cases for $\dfrac{\partial^2 \log L}{\partial \theta_{jk} \partial a_{ik}}$. There exist many settings of a_{ik} and θ_{jk} such that $\dfrac{\partial^2 \log L}{\partial \theta_{jk} \partial a_{ik}} \neq 0$. For example, if a random student j^* and ability $k^*$is sampled from the population, then $\mathbb{E}[\theta_{j^*k^*}] = 0$ (recall that student ability is assumed to be normally distributed across a population). Plugging in $\theta_{j^*k^*} =$

0 into Equation 2.29 gives

$$\frac{\partial^2 \log L}{\partial \theta_{jk} \partial a_{ik}}\Big|_{\theta_{jk}=0} = u_{ij}(1 - P_{ij}) - (1 - u_{ij})P_{ij} \neq 0$$

So we can see that the true Jacobian J of the gradient vector $\boldsymbol{f}$ does not actually have block diagonal structure, because the upper right and bottom left blocks contain nonzero entries. The assumption that items and examinees are independent of one another, while computationally necessary, limits the effectiveness and accuracy of JMLE.

Even with this assumption, the case of multidimensional latent traits still presents a computational burden. With J in block diagonal form, a single iteration of Equation 2.26 requires the inversion of N distinct $K \times K$ sub-matrices and n distinct $(K + 1) \times (K + 1)$ sub-matrices. To perform these operations in parallel, access to $N + n$ threads (a large amount of resources) is required.

In addition to computational issues, JMLE for IRT parameter estimation suffers from an identification problem for student ability parameters. The estimates produced by JMLE are only unique up to a linear transformation, and so an anchoring procedure is required in each Newton-Raphson iteration [4]. Each parameter estimate must be re-scaled via Gaussian normalization – for example, a student ability parameter is updated as $\hat{\theta}_{jkt} \leftarrow \frac{\theta_{jkt} - \overline{\hat{\theta}_{:kt}}}{\sigma(\hat{\theta}_{:kt})}$, where $\overline{\hat{\theta}_{:kt}}$ is the mean of all student's ability k at iteration t and $\sigma(\cdot)$ is the standard deviation.

JMLE also experiences difficulties in estimating the ability parameters for students who answer all items correctly or answer all items incorrectly. In this case, maximizing Equation 2.24 causes θ to become unbounded. Note that for a student

with a perfect score, increasing the estimate to their latent ability towards ∞ will increase the log-likelihood – likewise, decreasing a student's ability estimate towards $-\infty$ who answered all questions incorrectly will also result in an increase to the log-likelihood.

A similar phenomenon occurs for the parameters of items which all students answer correctly or all students answer incorrectly. For example, an item that all students answer incorrectly is usually interpreted as a rather difficult item. But JMLE takes this to the extreme, and the difficulty parameter is increased without bound in each iteration. It is straightforward to show that for an item i^* where $u_{i^*j} = 0$ for all j, the partial derivative of Equation 2.24 with respect to the difficulty parameter b_{i^*} of item i^* is strictly increasing:

$$\frac{\partial \log L}{\partial b_{i^*}} = \sum_{j=1}^{N} P_{i^*j} > 0 \tag{2.30}$$

2.4.1.1 Recent Adaptations

More recently, researchers have modified the JMLE method to overcome some of its flaws. Chen et al. [16] proposed constraining parameters to a feasible set in order to avoid the issue of unbounded parameter estimates. This results in the constrained optimization problem of maximizing the same log-likelihood in Equation 2.24, subject to the constraints

$$\sqrt{1 + ||\boldsymbol{\Theta}_j||_2^2} \leq C, \quad \sqrt{b_i^2 + ||\boldsymbol{a}_i||_2^2} \leq C, \quad \text{for all } 1 \leq i \leq n, \quad 1 \leq j \leq N \tag{2.31}$$

This optimization problem is solved iteratively via a projected gradient descent method – note that this is a first-order method – another simplification from

the second-order Newton-Raphson method described in Section 2.4.1. In general, a regular gradient descent iteration is defined by

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \nabla \mathcal{L}(\boldsymbol{x}_t) \tag{2.32}$$

where $\boldsymbol{x}$ are the variables to be optimized, $\mathcal{L}$ is the objective function, and η is the learning rate [84].

The projected gradient descent algorithm used by Chen et al. makes a small adjustment to Equation 2.32 to ensure that the iterates $\boldsymbol{x}_{t+1}$ remain in the feasible set. If $||\boldsymbol{x}_t - \eta \nabla \mathcal{L}(\boldsymbol{x}_t)||_2^2 \leq C$, then perform the usual update in Equation 2.32. But if $||\boldsymbol{x}_t - \eta \nabla \mathcal{L}(\boldsymbol{x}_t)||_2^2 > C$, then project back onto the feasible set:

$$\boldsymbol{x}_{t+1} = \frac{C}{||\boldsymbol{x}_t - \eta \nabla \mathcal{L}(\boldsymbol{x}_t)||_2^2} (\boldsymbol{x}_t - \eta \nabla \mathcal{L}(\boldsymbol{x}_t)) \tag{2.33}$$

The student and item parameters are updated in an alternating fashion. Given initial guess at $t = 0$ for the student parameters $\boldsymbol{\Theta}_{j,0}$ and item parameters $[\boldsymbol{a}_{i,0}, b_{i,0}]^\top$ for all students $1 \leq j \leq N$ and all items $1 \leq i \leq n$, each iteration is performed as

$$\begin{aligned} &\text{For each } j \text{ in parallel:} \\ &\quad \boldsymbol{\Theta}_{j,t+1} \leftarrow \boldsymbol{\Theta}_{j,t} - \eta \nabla \log L(\boldsymbol{\Theta}_{j,t}, [\boldsymbol{a}_{i,t}, b_{i,t}]^\top) \\ &\text{For each } i \text{ in parallel:} \\ &\quad [\boldsymbol{a}_{i,t+1}, b_{i,t+1}]^\top \leftarrow [\boldsymbol{a}_{i,t}, b_{i,t}]^\top - \eta \nabla \log L(\boldsymbol{\Theta}_{j,t+1}, [\boldsymbol{a}_{i,t}, b_{i,t}]^\top) \end{aligned} \tag{2.34}$$

Notice that when updating the item parameters in iteration $t + 1$ (the second line of Equation 2.34), the recently computed iterates $\boldsymbol{\Theta}_{j,t+1}$ are used.

The alternating scheme is similar to that proposed by Birnbaum [8] for JMLE for Newton-Raphson iterations in Equation 2.26. The two simplifications of con-

straining the feasible parameter space and using a first-order method fix the issue of unbounded parameter estimates and lessens the burden of estimating a large number of students.

However, notice that as the number of students increase, more threads are required to perform Equation 2.34 fully in parallel. In particular, the number of parameters to be updated via projected gradient descent increases linearly with the number of students and the amount of work for each thread depends linearly on the number of latent abilities.

The identifiability issue faced by JMLE is also experienced in CJMLE – estimates to student and item parameters are only unique up to a linear transformation. So after estimates have been obtained, they must be re-scaled so that among the population of students, $\boldsymbol{\Theta} \sim \mathcal{N}(0, I)$. Note that this assumes a student's K latent abilities are independent and not correlated with one another.

2.4.2 Marginal MLE via Expectation-Maximization

Another method of estimating IRT parameters utilizes the assumption that $\boldsymbol{\Theta}$ follows some prior distribution, usually $\mathcal{N}(0, I)$, while estimating parameters. This comes into play by applying the law of total probability to the observed data likelihood function in Equation 2.23 to arrive at the marginal likelihood function [4]:

$$\mathcal{M} = \prod_{j=1}^{N} P(\boldsymbol{u}_j) = \prod_{j=1}^{N} \int P(\boldsymbol{u}_j | \boldsymbol{\Theta}) g(\boldsymbol{\Theta}) d\boldsymbol{\Theta} \tag{2.35}$$

where $g(\boldsymbol{\Theta})$ is the prior distribution of student abilities.

Here, the student ability parameters are integrated out – replacing the assumed

point-estimates of student ability (as in JMLE) with a probability distribution over the student population. The goal then is to estimate the item parameters $\boldsymbol{a}_i$ and b_i by taking appropriate derivatives of $\log \mathcal{M}$. A notable difficulty is the presence of the integral in Equation 2.35 – this will be explored in Section 2.4.2.1

The high-level idea of Marginal Maximum Likelihood Estimation (MMLE) is to estimate the item parameters from using the observed student responses and knowledge of the prior distribution of student abilities. Though these student parameters are unobservable, the responses $\boldsymbol{u}_j$ can be used to make inferences about $\boldsymbol{\Theta}$; namely, the expectation of the joint distribution of $P(U, \boldsymbol{\Theta}|\boldsymbol{a}, \boldsymbol{b})$.

Using Bayes Theorem, write the posterior probability of student j's latent abilities as

$$P(\boldsymbol{\Theta}_j|\boldsymbol{u}_j) = \frac{P(\boldsymbol{u}_j|\boldsymbol{\Theta}_j)g(\boldsymbol{\Theta}_j)}{P(\boldsymbol{u}_j)} = \frac{P(\boldsymbol{u}_j|\boldsymbol{\Theta}_j)g(\boldsymbol{\Theta}_j)}{\int P(\boldsymbol{u}_j|\boldsymbol{\Theta})g(\boldsymbol{\Theta})d\boldsymbol{\Theta}} \tag{2.36}$$

Consider taking partial derivatives of $\log \mathcal{M}$ as described in Equation 2.35 with respect to an item parameter $x_i \in [\boldsymbol{a}_i, b_i]^\top$. Using the identity that $\frac{d}{dx}f(x) = f(x) \cdot \left[\frac{d}{dx}\log f(x)\right]$ for all continuously differentiable $f : \mathbb{R} \to (0, \infty)$ and Equation 2.36, we have

$$\begin{aligned}
\frac{\partial \log \mathcal{M}}{\partial x_i} &= \sum_{j=1}^{N} \frac{1}{P(\boldsymbol{u}_j)} \int \frac{\partial}{\partial x_i} \left[P(\boldsymbol{u}_j|\boldsymbol{\Theta})\right] g(\boldsymbol{\Theta})d\boldsymbol{\Theta} \\
&= \sum_{j=1}^{N} \frac{1}{P(\boldsymbol{u}_j)} \int P(\boldsymbol{u}_j|\boldsymbol{\Theta}) \cdot \frac{\partial}{\partial x_i} \left[\log P(\boldsymbol{u}_j|\boldsymbol{\Theta})\right] g(\boldsymbol{\Theta})d\boldsymbol{\Theta} \\
&= \sum_{j=1}^{N} \int \frac{P(\boldsymbol{u}_j|\boldsymbol{\Theta})g(\boldsymbol{\Theta})}{P(\boldsymbol{u}_j)} \cdot \frac{\partial}{\partial x_i} \left[\log P(\boldsymbol{u}_j|\boldsymbol{\Theta})\right] d\boldsymbol{\Theta} \\
&= \sum_{j=1}^{N} \int P(\boldsymbol{\Theta}|\boldsymbol{u}_j) \cdot \frac{\partial}{\partial x_i} \left[\log P(\boldsymbol{u}_j|\boldsymbol{\Theta})\right] d\boldsymbol{\Theta}
\end{aligned} \tag{2.37}$$

The particular values of $\frac{\partial \log P(\boldsymbol{u}_j|\boldsymbol{\Theta})}{\partial a_{ik}}$ and $\frac{\partial \log P(\boldsymbol{u}_j|\boldsymbol{\Theta})}{\partial b_i}$ can be found with some difficulty [4] to plug into the last line of Equation 2.37. The simplest case where $K = 1$ (unidimensional IRT) gives

$$\begin{aligned} \frac{\partial \log \mathcal{M}}{\partial a_{i1}} &= \sum_{j=1}^{N} \int_{\mathbb{R}} (\theta - b_i)(u_{ij} - P_{ij}) P(\theta|\boldsymbol{u}_j) d\theta \\ \frac{\partial \log \mathcal{M}}{\partial b_i} &= -a_{i1} \sum_{j=1}^{N} \int_{\mathbb{R}} (u_{ij} - P_{ij}) P(\theta|\boldsymbol{u}_j) d\theta \end{aligned} \tag{2.38}$$

Regardless, solving $\frac{\partial \log \mathcal{M}}{\partial x_i} = 0$ requires computing an integral over the latent ability $\boldsymbol{\Theta} \in \mathbb{R}^K$.

Solving this issue is tackled using quadrature and the Expectation-Maximization (EM) algorithm [28] [11]. Some modifications to the EM algorithm use Monte Carlo methods to compute the integral (MCEM) [60]. The K-dimensional integral is discretized into $H = c_1 \times c_2 \times \cdots \times c_K$ categories of ability levels. Then each student is categorized into one of X_h ability groups, $1 \leq h \leq H$

The EM algorithm is described by two steps which are repeated until convergence. Given initial guesses of the estimates $\hat{a}_{i1}^{(0)}$ and $\hat{b}_i^{(0)}$, perform for each iteration $t \geq 1$:

1. Expectation step:

   (a) Use quadrature (or Monte Carlo) to estimate the posterior probability $P(\boldsymbol{\Theta}_j|\boldsymbol{u}_j) \approx P(X_h|\boldsymbol{u}_j)$ for each student. Note that these values depend on the values $\hat{a}_{i1}^{(t-1)}$ and $\hat{b}_i^{(t-1)}$.

   (b) Compute the *expected* number of students which fall into each ability level X_h. This quantity can be written as $\overline{s}_h := \sum_{j=1}^{N} P(X_h|\boldsymbol{u}_j)$.

(c) Compute the *expected* number of correct responses to item i by students in ability level X_h, which can be written as $\overline{r}_{ih} := \sum_{j=1}^{N} u_{ij} P(X_h|\boldsymbol{u}_j)$.

2. Maximization step:

(a) The values computed in the expectation step provide an approximate form of Equation 2.38 (for the unidimensional case):

$$\begin{aligned}
\frac{\partial \log \mathcal{M}}{\partial a_{i1}} &\approx \sum_{j=1}^{N}\sum_{h=1}^{H}(X_h - b_i)(u_{ij} - P_{ih})P(X_h|\boldsymbol{u}_j) \\
&\approx \sum_{h=1}^{H}(X_h - b_i)(\overline{r}_{ih} - P_{ih}\overline{s}_h) \\
\frac{\partial \log \mathcal{M}}{\partial b_i} &\approx -a_{i1}\sum_{j=1}^{N}\sum_{h=1}^{H}(u_{ij} - P_{ih})P(X_h|\boldsymbol{u}_j) \\
&\approx -a_{i1}\sum_{h=1}^{H}(\overline{r}_{ih} - P_{ih}\overline{s}_h)
\end{aligned} \tag{2.39}$$

(b) Solve the optimization problem: find the roots of Equation 2.39, which serve as this iteration's estimates $\hat{a}_{i1}^{(t)}$ and $\hat{b}_i^{(t)}$.

After item parameters have been estimated, then student ability parameters can be estimated independently using MLE.

2.4.2.1 Curse of Dimensionality

Similar to JMLE, using the EM algorithm to perform MMLE faces difficulties with large datasets, though for a different reason. While JMLE requires inverting a large matrix, MMLE must compute many integrals which may be high-dimensional. As the number of latent traits K increases linearly, the number of points required to compute the integral in Equation 2.37 with respect to $\boldsymbol{\Theta}$ grows exponentially [14]. For example, the popular `mirt` package [15] sets the default number of quadrature points

per ability dimension at only 3 when $K \geq 6$. Yet this is still a problem; if $K = 10$, then $H = 3^{10} = 59,049$ quadrature points are used in each iteration. In Chapter 4, we analyze a synthetic dataset with $K = 20$ latent traits – the EM algorithm certainly cannot handle $H = 3^{20} = 3,486,784,401$ quadrature points.

This highlights one aspect of the curse of dimensionality in data science – as the number of features (latent traits) increases, performing accurate analysis becomes computationally infeasible. The specific problem in the case of MMLE relates to a large number of quadrature points (or sample points in MCEM) to compute high-dimensional integrals.

2.4.3 Metropolis-Hasting Robbins-Monroe

A more modern Bayesian approach to estimating IRT parameters was introduced by Li Cai [13, 14]. This method combines the Metropolis-Hastings sampling algorithm [45] with the Robbins-Monro stochastic optimization algorithm [79] to maximize the observed data likelihood. The Metropolis-Hastings Robbins Monro (MHRM) method is tailored for high-dimensional IRT models, and does not rely on numerical integration or sampling large volume spaces [43].

MHRM is the standard for estimating IRT parameters – it is the default method in software packages such as `mirt` [15]. While it was certainly a step forward in high-dimensional IRT parameter estimation, the method still faces difficulty when given a very large dataset – particularly, an increase in the number of students and number of items causes issues.

When trying to implement MHRM using `mirt` on large datasets, the amount of memory required to keep track of all parameters is multiple gigabytes, and the algorithm will not run. This is due to the fact that the MHRM algorithm still requires a computation of a large Jacobian matrix $J(\boldsymbol{x}) = \dfrac{\partial^2 \log L}{\partial x_i \partial x_j}$ where $\boldsymbol{x}$ is any parameter to be estimated, similar to in JMLE. After a stochastic approximation, a matrix of the same size must be inverted in each iteration.

2.4.4 Variational Methods

Very recently, variational inference approaches to IRT parameter estimation have gained attention. The high-level idea of variational inference is to treat difficult-to-compute probability density estimation as an optimization problem, and pick a distribution which most closely approximates the true posterior [10]. When applied to parameter estimation in IRT, this translates to learning a probability distribution, rather than directly maximizing individual parameters. So in contrast to the previously described techniques such as MLE and MHRM, variational inference methods do not directly optimize any student parameters $\boldsymbol{\Theta}$. In other words, there is no desire to solve $\dfrac{\partial \log L}{\partial \theta_{jk}} = 0$. ML2P-VAE, the primary work of this thesis introduced in Chapter 3, can be understood as a variational inference method for IRT.

Two new methods which are most closely related to this thesis also have ties to variational autoencoders, described in Section 2.2. Wu et al. [98] presents a new objective function, the variational IRT lower bound (VIBO). This approach assumes additional knowledge of the prior distribution of item parameters. This is used to

then add an additional KL divergence term to the VAE loss function in Equation 2.11. This approach is much faster than MLE approaches, and utilizes neural networks to parameterize the approximate distributions.

In a related application, Wang et al. [97] introduce VarFA, which uses variational inference in a factor analysis framework. Though factor analysis is not quite the same as parameter estimation, IRT can be considered a factor analysis model – an unknown student parameter explains how the student answers questions to an assessment. VarFA uses neural networks to approximate a variational distribution in order to speed up Bayesian inference. This approach is applicable to many factor analysis models (including IRT), and can be understood as a VAE with the decoder being an FA model.

CHAPTER 3
THE ML2P-VAE METHOD FOR IRT PARAMETER ESTIMATION

The primary contribution of this thesis is the development of a method for IRT parameter estimation which uses a modified VAE. The method, titled "ML2P-VAE", is interesting from multiple perspectives. In the application area, it is an unconventional approach that produces estimates with similar accuracy of traditional parameter estimation techniques. Further, ML2P-VAE scales much better than traditional methods as the number of latent abilities becomes large.

In traditional IRT parameter estimation, large datasets present a *burden* – they require more parameters to be estimated and a larger number of computations. But ML2P-VAE, through the lens of machine learning, views large datasets as a *blessing* – bigger data provides more information (more responses u_{ij}) to learn from in order to more accurately reconstruct inputs. A better reconstruction of input responses directly correlates with improved parameter estimates.

In the field of machine learning, ML2P-VAE is seen as an unsupervised learning technique which yields explainable results. A variational autoencoder is used in an unorthodox way – VAE are typically used as generative models where only the decoder is used at test-time. In ML2P-VAE, test-time involves feeding student responses through the encoder to obtain a prediction of latent traits – a result of an interpretable hidden neural network layer. The trainable parameters in the decoder are able to be understood in a real-world context – they serve as estimates to item parameters to the ML2P model from Section 2.3.4. In fact, the entire VAE decoder can be interpreted

as an approximate ML2P model (see Equation 2.18).

Another important contribution of this thesis is the development of a novel neural network architecture which fits a VAE to a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ where the latent code is correlated. This is a significant difference from previous implementations of VAE, which always assume that the dimensions of the latent code are independent of each other (see Equation 2.9). The more general architecture presented in this thesis is particularly helpful when additional domain knowledge of the latent code is accessible.

3.1 ML2P-VAE Method Description

Given an assessment with n items which tests K latent skills, assume that N students take this exam. Data is given as an $N \times n$ binary matrix U, as in Equation 2.12. No information about the student latent ability parameters $\boldsymbol{\Theta} \in \mathbb{R}^K$ or the item parameters a_{ik} and b_i is provided. However, assume that there is access to an expert-annotated binary Q-matrix detailing the item-skill associations, described by Equation 2.21. Implicitly, we assume that the student responses were generated by the ML2P model, introduced in Equations 2.18 and 2.22:

$$P(u_{ij} = 1 | \boldsymbol{\Theta}_j; \boldsymbol{a_i}, b_i) = \frac{1}{1 + \exp\left(-\sum_{k=1}^{K} q_{ik} a_{ik} \theta_{jk} + b_i\right)} \tag{3.1}$$

This assumption directly links IRT with VAE. Recall from Section 2.2.2 that the VAE decoder parameterizes the posterior distribution $p_\beta(\boldsymbol{x}|\boldsymbol{z})$, seen in Figure 2.2. Replace the observed data $\boldsymbol{x}$ with the student responses $\boldsymbol{u}_j$ and the latent code $\boldsymbol{z}$ with the latent traits $\boldsymbol{\Theta}_j$, and we see that Equation 3.1 provides an explicit form for the

posterior distribution $p_\beta(\boldsymbol{x}|\boldsymbol{z})$ learned by a VAE decoder. Recall that the subscript β references a particular setting of the weights in the VAE decoder, drawing parallel with the item parameters $\boldsymbol{a}_i$ and b_i. Though $\boldsymbol{a}_i$ and b_i are unknown, the constraints imposed by entries in the Q-matrix q_{ik} provide enough structure for the neural network to learn these values during the training process, so long as Q is sufficiently sparse [50].

A number of specifications are required in order to use a VAE as an IRT parameter estimation method. First, set up the neural network so that the input and output layers each have n nodes, each node representing one item. The inputs are the binary response vectors $\boldsymbol{u}_j$ defined in Equation 2.12 and the outputs are approximations of the probability of student j answering each item correctly.

The dimension of the hidden distribution (the output of the encoder) must be equal to K, the number of latent skills. The usual VAE loss function described in Equation 2.11 is still used to optimize ML2P-VAE. The KL-Divergence term acts as a regularizer between the posterior distribution produced by the encoder $q_\alpha(\boldsymbol{\Theta}_j|\boldsymbol{u}_j)$ and prior distribution of latent abilities $p^*_\theta(\boldsymbol{\Theta})$. Usually, these distributions are assumed to be independent Gaussian, but we extend this idea to a more general multivariate Gaussian distribution where the latent traits $\boldsymbol{\Theta}$ are correlated in Section 3.1.1.

The modifications to a typical VAE architecture are focused on the decoder. No hidden layers are used in the decoder. Instead, a non-dense matrix of weights connects the decoder input to the decoder output. The non-zero weights here are determined by the binary Q-matrix from Equation 2.21; recall that the input to the

decoder has K nodes, the decoder output has n nodes, and $Q \in \{0,1\}^{n\times K}$. So for a trainable weight w_{ik} in the VAE decoder, if $q_{ik} = 0$, then fix $w_{ik} = 0$ as well, and it will never be updated during the training process. This modification is key to allowing interpretation of the hidden latent distribution of a VAE, and is visualized in Figure 3.1.

The ML2P-VAE method requires the use of the sigmoid activation function

$$\sigma(z_i) = \frac{1}{1+e^{-z_i}} = \frac{1}{1+\exp\left(-\sum_{k=1}^{K} w_{ik}\alpha_k + \beta_i\right)} \tag{3.2}$$

in the output layer (see Appendix A.2). Here, $z_i = \sum_{k=1}^{K} w_{ik}\alpha_k + \beta_i$ is the input to the i-th node in the decoder, where w_{ik} is the weight between the k-th and i-th nodes in the decoder input and decoder output layer and β_i is the additive bias in the output layer. α_k is the activation of the k-th node in the decoder input layer, which is the sample drawn from the posterior distribution produced by the encoder.

Note the similarity between Equation 3.2 and the ML2P model in Equation 3.1. The constraint on the weights along with the sigmoid activation function allows for interpretation of the VAE decoder as an approximate ML2P model.

Specifically, the nonzero decoder weights w_{ik} can be interpreted as estimates to the discrimination parameters a_{ik}, the output bias β_i can be interpreted as estimates to the difficulty parameters b_i, and the activations α_k produced by the encoder (given the response input $\boldsymbol{u}_j$) can be interpreted as estimates to the student ability parameter θ_{kj}. All of this explainability is allowed by the constraint on the decoder weights matrix imposed by the binary Q-matrix.

Further modifications can improve the performance of ML2P-VAE. In IRT,

discrimination parameters are assumed to be non-negative, because an increase in a skill should never decrease the probability of answering an item correctly. With this assumption in mind, requiring all decoder weights $w_{ik} \geq 0$ avoids a potential identification problem, since $\theta^* \cdot a^* = (-\theta^*) \cdot (-a^*)$.

To summarize, the ML2P-VAE model takes as input the student responses $\boldsymbol{u}_j$ and maps them through a VAE encoder to estimates of the student's latent abilities $\boldsymbol{\Theta}_j$. This estimate is sent through a *modified* VAE decoder, whose trainable parameters serve as estimates to the ML2P model. The final output of the VAE decoder is the estimated probability of student j answering each item i correctly, $P_{ij} = \sigma(\boldsymbol{a}_i^\top \boldsymbol{\Theta}_j + b_i)$ as described in Equation 3.2, which is treated as a reconstruction of the inputs u_{ij}.

3.1.1 Full Covariance Matrix Implementation

In this section, we detail a novel architecture for a VAE which fits observed data to a latent space with correlated latent code, $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$. There are many publicly available code examples of VAE implementations which assume the latent space follows a standard normal distribution $\mathcal{N}(0, I)$. But it is not so common to train a VAE which assumes that the latent prior $p^*_\theta(\boldsymbol{\Theta})$ has correlated dimensions. Since most applications do not attempt to interpret hidden layers of a VAE, there is no available information on the correlations of abstract, unobservable features. Additionally, it can be beneficial to force the latent dimensions to be independent of one another when considering the usual applications of VAE.

In IRT, we may be able to quantify the correlation between latent abilities,

presenting need for the ML2P-VAE model to take advantage of this information. This task is nontrivial due to two mechanisms of the VAE:

(1) sampling from the learned distribution as in Equation 2.10, and

(2) calculating Kullback-Leibler Divergence as in Equation 2.9.

These two characteristics must be addressed when constructing a neural architecture which accounts for correlated $\boldsymbol{\Theta}$.

After training a VAE, sending a data point $\boldsymbol{u}_0$ through the encoder needs to give a set of values that correspond to a probability distribution. For a K-dimensional multivariate Gaussian distribution, these values are a vector $\boldsymbol{\mu}_0 \in \mathbb{R}^K$ and a symmetric, positive-definite matrix $\Sigma_0 \in \mathbb{R}^{K\times K}$. Sampling from $\mathcal{N}(\boldsymbol{\mu}_0, \Sigma_0)$ requires a matrix G_0 such that $G_0 G_0^\top = \Sigma_0$. This matrix factorization G_0 is not necessarily unique, but it can be convenient to use the Cholesky decomposition of Σ_0 where G_0 is lower triangular [3]. The sample from the multivariate Gaussian distribution is calculated as

$$\boldsymbol{z}_0 = \boldsymbol{\mu}_0 + G_0 \boldsymbol{\varepsilon_0} \tag{3.3}$$

where $\boldsymbol{\varepsilon_0} = (\varepsilon_1, \ldots, \varepsilon_K)^\top$and $\varepsilon_i \sim \mathcal{N}(0,1)$.

The KL-Divergence between two K-variate Gaussian distributions is given as

$$\begin{aligned} &\mathcal{D}_{KL}\left[\mathcal{N}(\boldsymbol{\mu}_0, \Sigma_0)\middle|\middle|\mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1)\right] = \\ &\frac{1}{2}\left(\mathrm{tr}(\Sigma_1^{-1})\Sigma_0) + (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)\Sigma_1^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) - K + \ln\left(\frac{\det\Sigma_1}{\det\Sigma_0}\right)\right) \end{aligned} \tag{3.4}$$

When using this in a VAE, $\mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1)$ corresponds to the prior $p^*_\theta(\boldsymbol{\Theta})$, and so $\boldsymbol{\mu}_1$ and Σ_1 are constant. Σ_1^{-1} only needs to be computed once, and this matrix inversion won't cause computation time problems at any point. Note that Equation 3.4 requires

computing $\ln \det \Sigma_0$, so we must have $\det \Sigma_0 > 0$ at all times during training. Recall that $\boldsymbol{\mu}_0$ and Σ_0 correspond to the input $\boldsymbol{u}_0$, and also depend on all the trainable weights and biases in the VAE encoder. These parameters are usually initialized randomly, and the user has little control over their values during training. If $\det \Sigma_0 \leq 0$ for any input $\boldsymbol{u}_0$ at any point while training, then it is not possible to compute the loss and gradient to perform backpropagation updates. Thus, a specific architecture which guarantees that $\det \Sigma_0 > 0$, regardless of the input $\boldsymbol{u}_0$ or encoder parameters, is required.

This architecture is described as follows. The input and output to the neural network consists of n nodes, each representing an item on an assessment which assesses K skills. Given an input $\boldsymbol{u}_0$ and after a sufficient number of hidden layers of sufficient size, the encoder outputs $K + \dfrac{K(K+1)}{2}$ nodes. The first K nodes represent the mean vector $\boldsymbol{\mu}_0$, and the remaining $\dfrac{K(K+1)}{2}$ nodes are arranged into a lower triangular matrix $L_0 \in \mathbb{R}^{K \times K}$. The covariance matrix is obtained using the matrix exponential $\Sigma_0 = e^{L_0} \cdot \left(e^{L_0}\right)^\top$.

Theorem 3.1. *Let L_0 be a lower triangular matrix, and define $\Sigma_0 = e^{L_0} \cdot \left(e^{L_0}\right)^\top$. Then Σ_0 is symmetric, positive-definite, and has positive determinant.*

Proof. Consider any lower triangular $L_0 \in \mathbb{R}^{K \times K}$. Define

$$G_0 = e^{L_0} = \sum_{n=1}^{\infty} \frac{L_0^n}{n!} = I + L_0 + \frac{1}{2} L_0 \cdot L_0 + \cdots$$

G_0 is lower triangular, since addition and multiplication of matrices preserve this property. Further, G_0 is nonsingular, because $\det G_0 = \det(e^{L_0}) = e^{\mathrm{tr} L_0} > 0$.

Set $\Sigma_0 = G_0 G_0^\top$. Clearly, Σ_0 is symmetric as $\Sigma_0^\top = (G_0 G_0^\top)^\top = G_0 G_0^\top = \Sigma_0$. Further,

$$\det \Sigma_0 = \det \left(G_0 G_0^\top \right) = \det G_0 \cdot \det G_0^\top = e^{\mathrm{tr} L_0} \cdot e^{\mathrm{tr} L_0} > 0.$$

So now for any nonzero $\boldsymbol{x} \in \mathbb{R}^K$,

$$\langle \Sigma_0 \boldsymbol{x}, \boldsymbol{x} \rangle = \boldsymbol{x}^\top \Sigma_0 \boldsymbol{x} = \boldsymbol{x}^\top G_0 G_0^\top \boldsymbol{x} = \langle G_0^\top \boldsymbol{x}, G_0^\top \boldsymbol{x} \rangle = ||G_0 \boldsymbol{x}||_2^2 > 0$$

Therefore, Σ_0 is positive-definite. □

Theorem 3.1 shows that in this specific neural network architecture, Σ_0 can be interpreted as a covariance matrix. Thus, the VAE encoder maps a data point $\boldsymbol{u}_0$ to a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_0, \Sigma_0)$. Additionally, the sampling operation in Equation 3.3 and KL-Divergence calculation in Equation 3.4 can always be carried out without issue.

A visualization of the ML2P-VAE architecture for correlated latent traits is shown in Figure 3.1. At test time, the estimate of the latent skills of a student with responses $\boldsymbol{u}_0$ is obtained via $\boldsymbol{\Theta}_0 = \boldsymbol{\mu}_0$. The output reconstruction $\boldsymbol{P}_0 = (P_{10}, P_{20}, \ldots, P_{n0})^\top$ refers to the reconstruction $\hat{\boldsymbol{u}}_0$ of the input $\boldsymbol{u}_0$, also interpreted as the probability of student 0 answering each question correctly:

$$P_{i0} = P(u_{i0} = 1 | \boldsymbol{\Theta}_0; \boldsymbol{a}_i, b_i)$$

3.1.2 Variants of ML2P-VAE

We consider three scenarios for using ML2P-VAE in practice: (a) the best case scenario where the covariance matrix between all latent traits is fully known, (b)

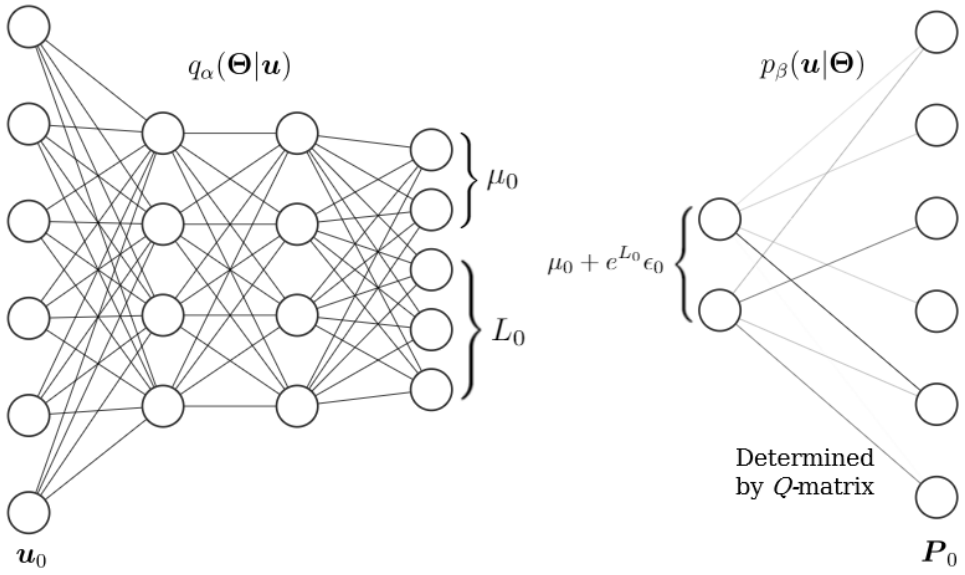

Figure 3.1: Visualization of the ML2P-VAE architecture for two correlated latent traits and six input items. Note that the trainable weights matrix in the decoder is not dense, but is determined by the given Q-matrix.

the exact covariance matrix unknown, so it is estimated using other methods, and (c) we simply assume that all traits are independent. These three situations result in three variations of the ML2P-VAE method: ML2P-VAE$_{full}$, ML2P-VAE$_{est}$, and ML2P-VAE$_{ind}$, respectively.

In order to estimate the correlations between latent traits for use of ML2P-VAE$_{est}$ in scenario (b), the student response matrix $U \in \mathbb{R}^{N\times n}$ is multiplied by the Q-matrix $Q \in \mathbb{R}^{n\times K}$. Denote $M = UQ \in \mathbb{R}^{N\times K}$. Then the Pearson correlation of the columns of M produce an approximate correlation matrix $\hat{\Sigma} \in \mathbb{R}^{K\times K}$, where each

entry $\hat{\sigma}_{kl}$ gives the approximate correlation between latent traits k and l.

$$\hat{\sigma}_{kl} = \frac{\sum_{i=1}^{N}(k_i - \bar{k})(l_i - \bar{l})}{\sqrt{\sum_{i=1}^{N}(k_i - \bar{k})^2}\sqrt{\sum_{i=1}^{N}(l_i - \bar{l})^2}} \tag{3.5}$$

where $\bar{k}$ and $\bar{l}$ are the mean values of the k-th and l-th columns of M, respectively.

A final variation of ML2P-VAE comes when the IRT model to be estimated is changed. If assuming that student responses are generated according to the Rasch model in Equation 2.15 rather than the ML2P model from Equation 2.18, then another variation of VAE parameter estimation methods can be considered. A more appropriate name for this alternative estimation method is Rasch-VAE.

Since there are no discrimination parameters in the Rasch model, only item difficulties and student abilities need to be estimated. To account for this, the weights in the VAE decoder are restricted to be *equal* to the Q-matrix, i.e. $w_{ik} = q_{ik}$. This still allows for interpretation of the learned distribution of the VAE as estimates to student abilities $\boldsymbol{\Theta}$, while requiring "discrimination parameters" to be equal to either one or zero, fitting more closely to Equation 2.15.

3.2 ML2Pvae Software Package for R

The ML2P-VAE method for parameter estimation has been compiled in an easy-to-use software package for R [19]. This allows researchers who may not have experience with neural networks to implement ML2P-VAE methods on a data set of their choosing. The package **ML2Pvae** is available on the Comprehensive R Archive Network (CRAN) and can be easily installed using the R command:

```
install.packages('ML2Pvae')
```

3.2.1 Package Functionality

ML2Pvae uses Tensorflow and Keras to build and train neural networks, but no knowledge of these libraries are required in order to use **ML2Pvae**. The package exports five functions available to the user. Two of these are used to construct Keras models, with optional parameters specifying the architecture of the neural network. The only parameters which require input from the user are the number of items on the exam, the number of latent abilities that the exam assesses, and the Q-matrix relating items and abilities.

The optional inputs in the model construction include a covariance matrix for latent traits, allowing for correlated skills and the implementation described in Section 3.1.1. An important feature for model selection gives the choice of the number of item parameters to use in the logistic IRT model. Though the package is called **ML2Pvae** for the Multidimensional Logistic 2-Parameter model, the package allows for estimating parameters with the Rasch model, so that Rasch-VAE from Section 3.1.2 can be implemented. In this case, there is only a difficulty parameter for each item; each discrimination parameter is fixed to be equal to 1. Other options when building ML2P-VAE models specify the number, size and activation functions of the hidden layers in the encoder.

Using the Keras models returned by the construction functions, **ML2Pvae** provides a function that can be used to train the VAE on data. This function acts as a wrapper for the `fit()` method in the Keras package. The final two methods obtain item parameter estimates and student ability parameter estimates. This is done by

grabbing the correct weights/biases from the decoder and feeding student responses through the encoder, respectively. A more detailed description of the functionality of **ML2Pvae**, along with a code tutorial, is given in Appendix B.

CHAPTER 4
ML2P-VAE RESULTS AND DISCUSSION

The ML2P-VAE method has been used in a various settings in multiple publications related to educational measurement [18, 21, 24]. The first paper, "Interpretable Variational Autoencoders for Cognitive Models" (*International Joint Conference on Neural Networks* 2019), introduced the ML2P-VAE method and gives some preliminary results on a small simulated data set. The second, "Autoencoders for Educational Assessment" (*Conference on Artificial Intelligence in Education* 2019), displays the advantages that a VAE holds over a regular autoencoder in the task of parameter estimation. The final publication, "Estimation of Multidimensional Item Response Theory Models with Correlated Latent Variables using Variational Autoencoders" (*Machine Learning* 2021), compares different variations of ML2P-VAE with traditional parameter estimation methods on both real and simulated data sets of various sizes, along with introducing the novel VAE architecture described in Section 3.1.1.

In this chapter, we first summarize all datasets used in experiments, then present all results from each publication. Finally, we explore future extensions of the ML2P-VAE method and summarize the contributions of this research.

4.1 Description of Data Sets

Sim-ECPE

This simulated data set is designed to mirror the real-world Examination for the Certificate of Proficiency in English, detailed further in the next dataset description. Sim-ECPE has 28 items assessing 3 latent traits. Values for the item parameters in the ML2P model were generated from a uniform distribution so that $a_{ik} \in [0.25, 1.75]$ and $b_i \in [-3, 3]$. The range for the discrimination parameters was chosen such that $0.25 \leq MDISC_i \leq 1.75$ for all i (see Equation 2.19). Up to 10,000 student abilities $\boldsymbol{\Theta} \in \mathbb{R}^3$ were sampled from $\mathcal{N}(0, I)$, and experiments are performed using different numbers of students N. Note that in Sim-ECPE, it is assumed that the latent traits are independent. We use a Q-matrix consistent with previous literature [25, 92, 46].

ECPE

The Examination for the Certificate of Proficiency in English (ECPE) is an exam with 28 items designed to certify learners of the English language. The set of responses we use is available in the **CDM** package for R [81]. This includes 2,922 students and a Q-matrix for three skills - "morphosyntactic rules", "cohesive rules", and "lexical rules". Since this is a real-world data set, there are not "true" values of item or student ability parameters to compare with the model estimates, so we must compare our results with previous literature.

Sim-6

A moderately-sized simulated data set, Sim-6 has 50 items evaluating 6 latent traits. The Q-matrix is also generated randomly, where each entry q_{ik} is sampled from Bern(0.2). To ensure each item requires at least one latent ability, if a row $q_{i:} = 0$ after sampling, then one random element in the row is changed to a 1. The discrimination parameters are chosen so that $a_{ik} \in [0.1, 1.3]$ and $b_i \in [-3, 3]$ and that $0.25 \leq MDISC_i \leq 1.75$. Abilities $\boldsymbol{\Theta} \in \mathbb{R}^6$ of 20,000 students were sampled from $\mathcal{N}(0, \Sigma)$, where Σ is a correlation matrix with all positive values generated using the SciPy package [96].

Sim-20

This large data set is generated in a similar manner to Sim-6, but includes 50,000 students, 200 items, and 20 latent traits. The Q-matrix was generated in the same way as that of Sim-6, where $q_{ik} \sim$ Bern(0.2). The difficulty parameters were sampled uniformly so that $b_i \in [-3, 3]$, and the discrimination parameters were sampled uniformly so that $a_{ik} \in [0.1, 0.9]$. As in Sim-6, the 20 latent abilities are correlated with one another (i.e. $\boldsymbol{\Theta} \sim \mathcal{N}(0, \Sigma)$), and the correlation matrix is generated in the same manner as in Sim-6.

Sim-4

Sim-4 contains 3,000 student responses to an exam with 27 items over 4 latent abilities. Another simulated dataset, the covariance matrix and Q-matrix were chosen more deliberately than the previous two datasets. Of the 4 skills in the correlation

matrix, one of them is entirely independent of the other three. The other three latent abilities had correlations of 0.25, 0.1, and 0.15 between them. These correlation values are much smaller than those of Sim-6 or Sim-20, resulting in a covariance matrix that is closer to the identity. The Q-matrix was chosen so that it contained 16 simple items (items requiring only one skill), 6 items requiring 2 latent abilities, 4 items requiring 3 latent abilities, and one item requiring all 4 skills. In this way, each of the possible $\binom{4}{k}$ combinations is present in the Q-matrix, for $k \in \{1, 2, 3, 4\}$.

4.2 Quantitative Results

4.2.1 Preliminary Results

This section describes the results presented at IJCNN 2019 [24], when the ML2P-VAE model was initially proposed. All experiments were performed on the Sim-ECPE dataset.

The latent traits values were generated independently from a $\mathcal{N}(0, I)$ distribution to compose three datasets with different sample sizes (N): 500, $5,000$, and $10,000$ subjects. All figures included in this section were generated using the largest set $N = 10,000$. The smaller sample size was fixed with the goal of comparison to previous results presented in the literature for the ML2P model, which use the traditional estimation process MCMC. The current results can be directly compared to the ones published by da Silva et al. [25], since the simulation scenarios are the same. The other two sample sizes were chosen to study the improvement of the estimation accuracy for the proposed model when provided larger sets of data.

Given a subject's latent traits $\boldsymbol{\Theta}_j$, we generate ten different replicates of responses using the ML2P-Q method [25] described in Equation 2.22. After we have ten sets of responses for N subjects, we train ten separate instances of our model and use the learned parameters from these networks to estimate $\hat{\boldsymbol{a}}$, $\hat{\boldsymbol{b}}$, and $\hat{\boldsymbol{\Theta}}$.

The architecture of the variational autoencoder, implemented using TensorFlow, is as follows: The encoder has an input layer of 28 nodes (one for each item), a hidden layer with 10 nodes, and outputs the distribution of the 3 latent traits, requiring 6 nodes. As mentioned in Section 3.1, the decoder has no hidden layers, and simply maps the 3 latent traits to the 28 items. The nonzero weights of the decoder are determined by the Q-matrix. Further, these weights are restricted to be non-negative and we use a sigmoidal activation function, which lines up with how the data was simulated with ML2P-Q.

Because of this similarity, we are able to interpret the weights and biases in the decoder of the VAE as estimates of the IRT parameters in Equation 3.1 used to generate the data. Though the encoder is still a black box, the VAE is constructed in a way so that we can interpret the weights and biases in the decoder.

In the left plot of Figure 4.1, we can see a clear correlation between the discrimination parameters used to generate the dataset and the weights in the decoder. There is an even stronger correlation between the difficulty parameters and the biases in the output layer, as seen in the right plot of Figure 4.1.

Estimates of the discrimination and difficulty parameters yield an improved interpretation of the neural network. The estimates for a_{ik} can be used to quantify

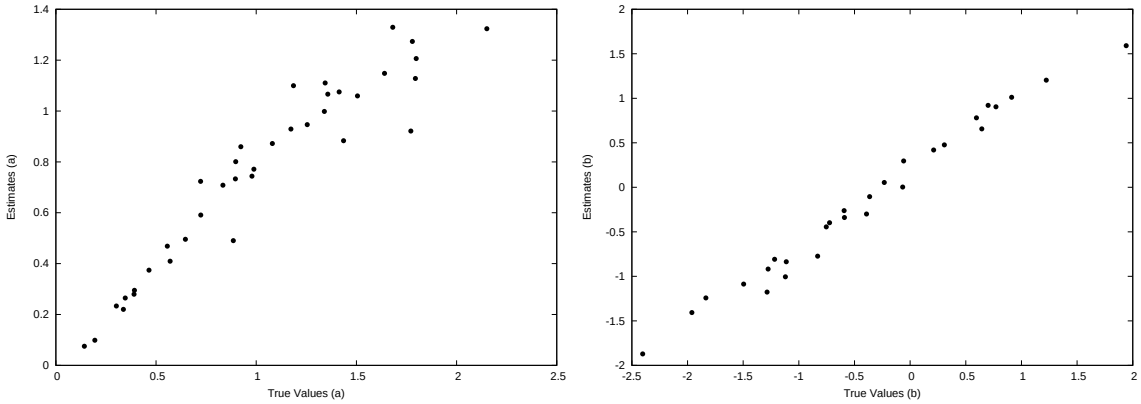

Figure 4.1: True versus estimated values for discrimination parameters (left) and difficulty parameters (right) with sample size 10,000.

the ability of an item to discriminate between individuals with different levels of knowledge. Smaller values correspond to an item with little discriminatory power and larger values correspond to an item that can easily discriminate between individuals with different levels of knowledge. Similarly, the estimates of b_i allow quantification of the difficulty of an item, again with smaller values corresponding to an easier item and larger values corresponding to a more difficult item.

We use root mean square error (RMSE), correlation (CORR), and absolute value of the relative bias (AVRB) between the estimated and true values of the parameters to evaluate the accuracy of parameter estimates. The absolute value of the relative bias (AVRB) was defined as:

$$\text{AVRB}_i = \left| \frac{\left(\frac{1}{10} \sum_{r=1}^{10} \hat{\lambda}_{ir} \right) - \lambda_i}{\lambda_i} \right|, \tag{4.1}$$

where λ_i refers to one of the true parameters of the model (a_{ik}, b_i, or θ_{jk}) and $\hat{\lambda}_{ir}$

refers to the respective estimate for data replicate r. These are shown in Table 4.1. As we increase the sample size of the dataset, we see an increase in correlation for all parameters. The error measures of the estimates to a_{ik} decrease, as is expected. Oddly, the error measures for b_i increase as the number of students increases - this issue is left for future work. However, this does not seem to affect the correlation between true and estimated b_i.

In general, it can be noted that while there is high correlation between true and estimated item parameters, the RMSE values are not particularly low. This is likely because the scale of the estimates is off – though the true values $b_i \in [-3, 3]$, the estimates obtained were significantly smaller, $\hat{b}_i \in [-1, 1]$, with one exception. So the difference between true and estimated values is sizable, even though they exhibit a strong linear relationship.

The encoder of our VAE also holds predictive power. Given a subject's assessment results, we feed this information forward through the encoder and return a prediction of that subject's latent traits. In Figures 4.2, 4.3, and 4.4, we observe an explicit relationship between the learned distributions $\hat{\theta}_k$ of the VAE and the latent traits θ_k for $k \in \{1, 2, 3\}$. The plots seem to have a sigmoidal tendency, rather than linear. This is not ideal, as it causes our model to struggle with accurately predicting the latent traits of subjects who have either very high or very low latent trait values.

The accuracy of these estimates is shown in Figure 4.5. Nearly 90% of the individuals have the absolute values of the difference between estimates and true values

AVRB				
Size	$\boldsymbol{a}_{:1}$	$\boldsymbol{a}_{:2}$	$\boldsymbol{a}_{:3}$	$\boldsymbol{b}$
500	0.779	0.699	0.759	1.188
5,000	0.539	0.281	0.585	1.673
10,000	0.284	0.159	0.264	1.894

RMSE				
Size	$\boldsymbol{a}_{:1}$	$\boldsymbol{a}_{:2}$	$\boldsymbol{a}_{:3}$	$\boldsymbol{b}$
500	0.976	0.931	0.850	1.038
5,000	0.587	0.823	0.414	1.494
10,000	0.322	0.346	0.264	1.670

CORR				
Size	$\boldsymbol{a}_{:1}$	$\boldsymbol{a}_{:2}$	$\boldsymbol{a}_{:3}$	$\boldsymbol{b}$
500	0.457	0.547	0.381	0.987
5000	0.779	0.710	0.990	0.982
10000	0.924	0.920	0.986	0.990

Table 4.1: Error measures for item parameter estimates.

under 0.5, which improves upon the results of the traditional estimation methods in MIRT [25], which achieved approximately 75% of the individuals with absolute bias under 0.5. The outliers (points outside the box) in Figure 4.5 are the left and right parts of the sigmoid shapes seen in Figures 4.2, 4.3, and 4.4.

4.2.1.1 Remarks

One drawback of the proposed model is that it requires a much larger sample size to obtain comparable results. However, despite the larger sample size, the running time of our neural network method is 40 times smaller (17 seconds, using our VAE model, versus 662 seconds, using MCMC), despite a larger sample size. As mentioned in Chapter 3, while large datasets present a burden for traditional parameter

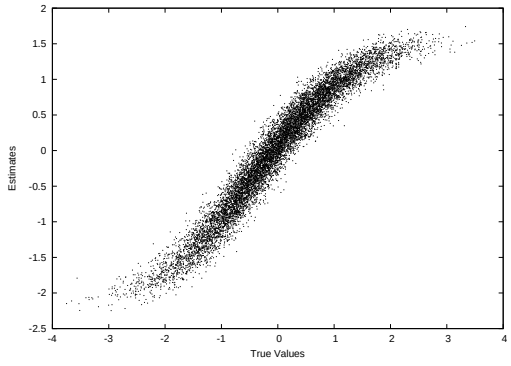

Figure 4.2: $\hat{\theta}_1$ estimates for the first latent variable.

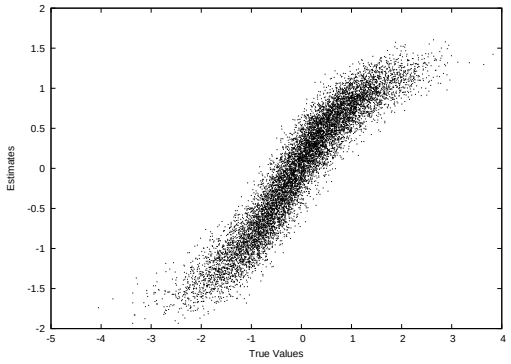

Figure 4.3: $\hat{\theta}_2$ estimates for the second latent variable.

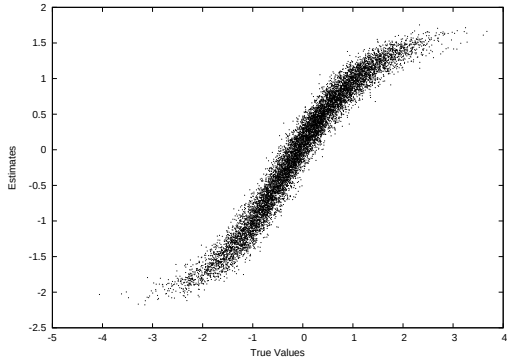

Figure 4.4: $\hat{\theta}_3$ estimates for the third latent variable.

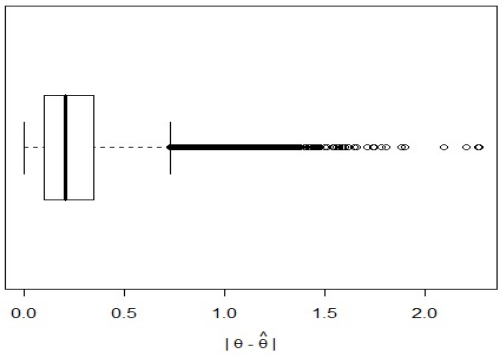

Figure 4.5: Absolute differences between true values and estimates of $\boldsymbol{\Theta}$.

estimation techniques, more data is beneficial to ML2P-VAE.

The latent trait estimates of the proposed VAE are highly correlated with the true values and are precisely estimated, except in the tails of the distribution. The results for the extreme latent values may be improved by increasing the number of items with difficulty parameter values around these regions. Because the latent trait and difficulty parameters have the same scale, increasing the amount of observed

information in the tails will improve the estimation results around that region.

Another contribution of this work is to present the relation between a common multidimensional IRT model and VAE. Using the VAE formulation to estimate ML2P parameters solves an important limitation faced by the traditional estimation processes: big data. Until now, the point and standard error estimation of latent trait parameters from MIRT models with medium to large dimension was infeasible, described in Section 2.4.

The simulation results demonstrate that the accuracy of the method improves with increased sample size. For samples sizes above 10,000, the parameter estimates of the items (weights bias of the decoder) and latent features are highly correlated with the actual values. Conversely, for smaller sample sizes ($N = 500$), traditional estimation methods (MCMC and MLE) give better results. For example, da Silva et al. [25] find good estimates with smaller sample sizes for the same experimental situation using MCMC, but with a running time around $40\times$ longer for $N = 500$, and $108\times$ longer for $N = 1000$, when compared to our VAE model.

Therefore, for situations with a large sample size and/or running time restrictions, the proposed VAE model surpasses traditional methods of IRT parameter estimation. Moreover, it is a promising way to overcome the computational infeasibility for high latent trait dimensions in IRT models, further explored in Section 4.2.3. It is an issue that should be considered for future studies. This conclusion is supported by scientific researchers in statistics [10].

4.2.2 Variational Autoencoder vs Regular Autoencoder in IRT

Shortly after introducing the ML2P-VAE method, comparisons between a variational autoencoder (VAE) and a regular autoencoder (AE) for parameter estimation were presented at AIED 2019 [18]. Guo et al. proposed a neural network approach to estimating student mastery in 2017 [39]. This neural network had auto-encoding structure, but was geared towards CDM and did not make a direct connection to IRT or parameter estimation. In this section, we empirically show that using a VAE produces better item and ability estimates than a regular autoencoder and analyze the differences in models leading to this improvement.

For these experiments, the same simulated dataset presented in Section 4.2.1, Sim-ECPE, is used here. The neural architecture used for all experiments includes 28 input/output nodes (one for each item), one hidden layer in the encoder with 10 nodes, and an encoded dimension of 3, representing three latent traits. The decoder has no hidden layers, with connections determined by a given Q-matrix. Of course, the VAE includes three extra nodes in the encoder output representing variance of each latent trait so that the VAE encoder produces a standard normal distribution, while the regular AE only has 3 nodes outputted by the encoder.

Three error measures for VAE and AE estimates are given in Table 4.2 and Table 4.3. These include absolute value relative bias (AVRB), root mean square error (RMSE) and Pearson correlation (CORR). The statistics for item parameter estimates in Table 4.2, where $\boldsymbol{a}_{:k}$ denotes the average measure taken over all items related to latent trait θ_k, and $\boldsymbol{b}$ is the average measure taken over all item difficulty

Model	$\boldsymbol{a}_{:1}$	$\boldsymbol{a}_{:2}$	$\boldsymbol{a}_{:3}$	$\boldsymbol{b}$	Statistic
AE	0.680	0.227	0.529	2.305	AVRB
VAE	0.284	0.159	0.264	1.894	
AE	0.585	0.481	0.534	1.651	RMSE
VAE	0.322	0.346	0.264	1.670	
AE	0.529	0.547	0.748	0.917	CORR
VAE	0.924	0.920	0.986	0.990	

Table 4.2: Error measures for item parameter recovery of AE and VAE.

parameters. Note that the AVRB values for difficulty parameters is rather high, likely due to some of the true values of b_i are very near zero (see Equation 4.1).

The item parameter estimates from VAE outperform those from AE for each category and measure. This is corroborated by the correlation plots in Figure 4.6 and Figure 4.7. The discrimination parameters are estimated much more consistently by a VAE, while there are many outlier estimates produced by the regular autoencoder. When looking at the difficulty parameter estimates, there seems to be a much tighter error bound on the VAE estimates than the AE estimates.

Results for student ability parameter estimates are shown in Table 4.3 and Figure 4.8. Again, we see that the error measures from VAE estimates are much lower than those from a regular AE. However, the correlation values are slightly better for AE, though the difference is not visible in the correlation plot. The reason that AE has poor AVRB and RMSE measures is because the ability parameter estimates are on a different scale than the true values. Notice in the left plot of Figure 4.8 that the vertical axis reaches over 4, while the vertical axis of right plot has closer range to the true scale of $\boldsymbol{\Theta}$. This is likely due to the KL-divergence term in the VAE loss

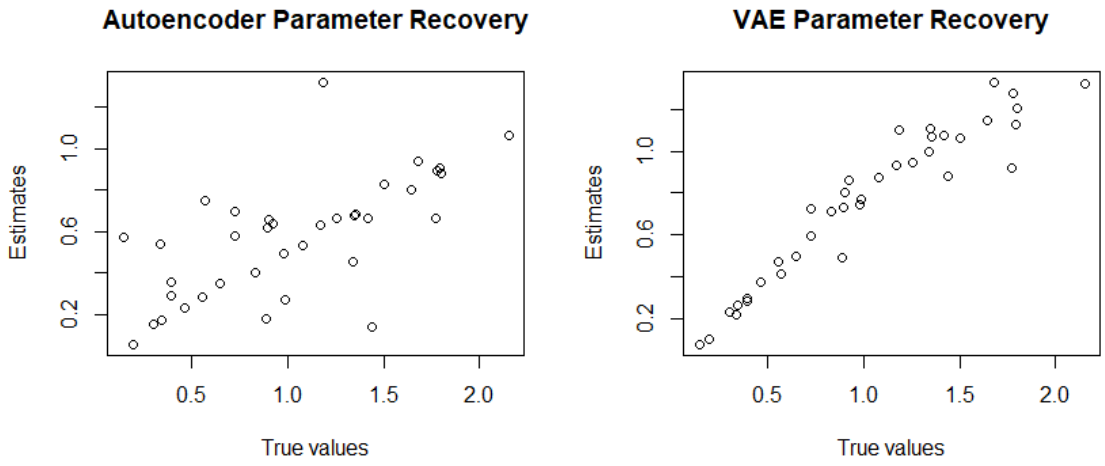

Figure 4.6: Autoencoder and VAE discrimination parameter (a_{ik}) recovery.

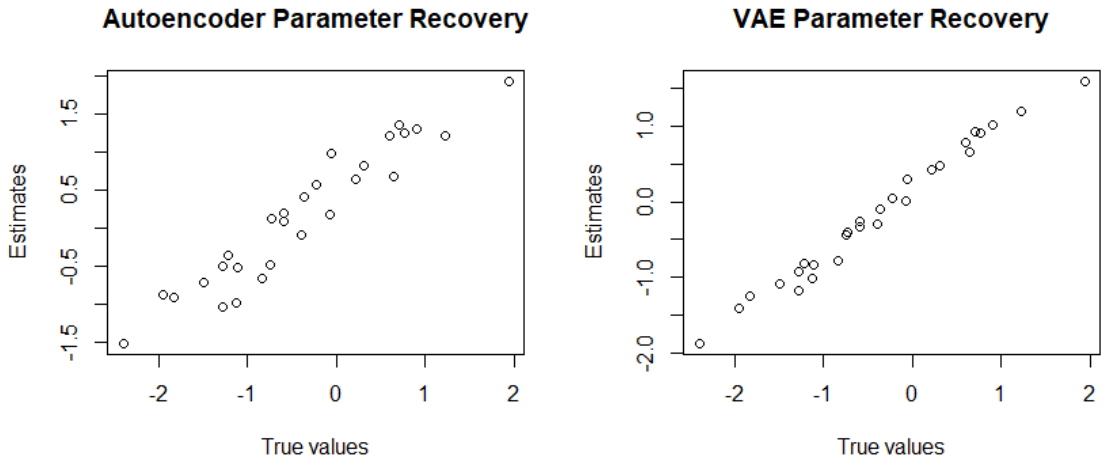

Figure 4.7: Autoencoder and VAE difficulty parameter (b_i) recovery.

function, better controlling the scale of the latent distribution to be near $\mathcal{N}(0, I)$.

The lack of a KL-divergence term in a regular AE also helps explain the poor discrimination parameter estimates shown in the right plot of Figure 4.6. The ML2P

Model	θ_1	θ_2	θ_3	Statistic
AE	7.425	3.107	16.260	AVRB
VAE	1.844	1.713	4.009	
AE	1.788	1.523	1.746	RMSE
VAE	0.664	0.760	0.646	
AE	0.970	0.937	0.971	CORR
VAE	0.965	0.940	0.969	

Table 4.3: Error measures for latent trait prediction of AE and VAE.

model can suffer from an identifiability issue without the assumption that student ability parameters follow some probability distribution [40]. Fixing this problem often requires an anchoring or rotation procedure after estimation [4]. For a VAE, this is avoided by incorporating the KL-divergence term in Equation 2.11 between the encoder output and the prior distribution $p^*_\theta(\mathbf{\Theta}) = \mathcal{N}(0, I)$.

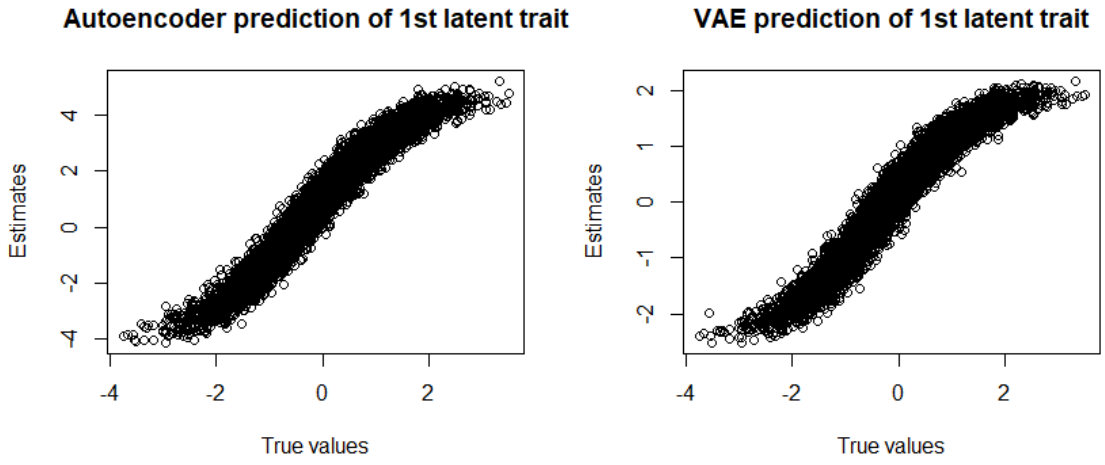

Figure 4.8: Autoencoder and VAE predictions for θ_1.

Both autoencoders and variational autoencoders can be used as IRT parameter estimation methods when a Q-matrix restricts weights in the decoder. In either case, adding interpretability to neural networks is interesting, but a VAE is able to incorporate an extra piece of domain knowledge in the prior distribution of $\boldsymbol{\Theta}$, leading to more accurate estimates. The results presented in this section demonstrate the advantage that the ML2P-VAE method holds over the regular autoencoder architecture proposed by Guo et al. [39].

4.2.3 ML2P-VAE vs Traditional Methods

In this section, a direct comparison of ML2P-VAE with traditional parameter estimation techniques for IRT. Three variants of ML2P-VAE are used: ML2P-VAE$_{full}$, ML2P-VAE$_{est}$, and ML2P-VAE$_{ind}$ as described in Section 3.1.2. These are compared against Metropolis-Hastings Robbins-Monro (MHRM) [13], Quasi Monte-Carlo Expectation Maximization (QMCEM) [15], and Monte-Carlo Expectation Maximization (MCEM) [11]. This work has been published in the journal *Machine Learning* [21].

A summary of each method's performance is given in Table 4.4. All experiments were conducted using Tensorflow for R on a laptop computer with a 2.9 GHz Intel Core i7-7500U CPU. Note that although artificial neural networks benefit greatly from GPU, we do not utilize this hardware and still get a sizable speedup in comparison to other methods. The results from traditional methods were obtained using default settings of the MIRT package [15].

In all variations of ML2P-VAE, we train the neural network with the Adam optimizer [52] for 10 epochs and batch size 1 (pure stochastic gradient descent). The specific encoder architecture of the neural network was dependent on the size of the data set. Sim-6 used two hidden layers of size 32 and 16, ECPE used two hidden layers of 16 and 8 nodes, and Sim-20 utilized two hidden layers of size 64 and 32. In each network, a sigmoid activation function was used in the encoder hidden layers and a linear activation function in the encoded distribution. As described earlier, the ML2P-VAE model requires the use of a sigmoidal activation function in the output layer of the decoder.

Note that when comparing error measures in Sim-6, the ML2P-VAE methods are competitive with traditional methods. In particular, assuming full knowledge of the latent trait covariances in ML2P-VAE yields discrimination, difficulty, and ability parameter estimates of similar accuracy to MHRM. When the assumption of known latent trait correlation is relaxed, the accuracy of parameter estimates understandably slip.

Although the ML2P-VAE methods are slightly less accurate than MHRM, they are much faster than traditional methods, especially as the number of latent traits increase. Much of this speedup is due to the fact that neural networks do not require numerical integration over the latent abilities, and instead use simple hidden neural layers. While quadrature or MCMC methods become infeasible on data sets any larger than Sim-6, this is no cause for concern with ML2P-VAE. Note that for neural networks of this size (50-200 inputs and latent dimension 6-20), the longer runtime is

Data Set	Method	$\boldsymbol{a}$.RMSE	$\boldsymbol{a}$.BIAS	$\boldsymbol{a}$.COR	$\boldsymbol{b}$.RMSE	$\boldsymbol{b}$.BIAS	$\boldsymbol{b}$.COR	$\boldsymbol{\Theta}$.RMSE	$\boldsymbol{\Theta}$.BIAS	$\boldsymbol{\Theta}$.COR	Runtime
(i) 6 abilities Sim-6	MHRM	0.0693	0.0319	0.9986	0.0256	-0.0021	0.9999	0.714	-0.0033	0.7006	1110s
	QMCEM	0.149	-0.067	0.9939	0.0376	-0.002	0.9998	0.7206	0.0023	0.6939	322s
	MCEM	0.1497	-0.0633	0.9936	0.0383	0.0035	0.9997	0.7206	-0.0016	0.6938	1009s
	ML2P-VAE$_{full}$	0.0705	0.0255	0.9985	0.0471	-0.0079	0.9996	0.6649	-0.0178	0.7476	343s
	ML2P-VAE$_{est}$	0.1803	0.0871	0.9891	0.064	-0.0131	0.9993	0.7109	0.0772	0.7082	364s
	ML2P-VAE$_{ind}$	0.1218	-0.0004	0.9944	0.0597	-0.0145	0.9994	0.7222	0.0316	0.6928	252s
(ii) 3 abilities ECPE	MHRM*	0*	0*	1*	0*	0*	1*	0*	0*	1*	162s
	QMCEM	0.0159	0.0035	0.9999	0.0067	-0.0005	1	0.0111	0.0007	0.9999	33s
	MCEM	0.0228	0.0148	0.9998	0.0064	-0.0008	1	0.0132	0.0026	0.9998	192s
	ML2P-VAE$_{full}$	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	ML2P-VAE$_{est}$	0.2794	0.2152	0.9713	0.148	0.0951	0.993	0.443	-0.0628	0.8237	61s
	ML2P-VAE$_{ind}$	0.3208	0.2184	0.9504	0.154	0.0872	0.9932	0.3063	0.01	0.9017	49s
(iii) 20 abilities Sim-20	MHRM	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	QMCEM	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	MCEM	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	ML2P-VAE$_{full}$	0.078	0.0473	0.9983	0.0608	0.0054	0.9996	0.6145	0.0065	0.7893	1292s
	ML2P-VAE$_{est}$	0.2992	-0.1304	0.9822	0.1655	0.1215	0.9987	0.7364	-0.0276	0.7257	961s
	ML2P-VAE$_{ind}$	0.2043	0.0592	0.9792	0.0958	-0.0029	0.9992	0.7054	0.0747	0.7135	850s
(iv) 4 abilities Sim-4	MHRM	0.0953	-0.0158	0.9966	0.0614	-0.0101	0.9988	0.6325	0.0118	0.7697	94s
	QMCEM	0.0938	-0.0160	0.9967	0.0614	-0.0179	0.9989	0.6326	0.0154	0.7696	29s
	MCEM	0.0951	-0.0138	0.9966	0.0644	-0.0199	0.9987	0.6326	0.0150	0.7696	196s
	ML2P-VAE$_{full}$	0.1326	0.0780	0.9960	0.0872	-0.0311	0.9978	0.6384	0.0210	0.7648	37s
	ML2P-VAE$_{est}$	0.2526	0.2106	0.9883	0.1035	-0.0337	0.9980	0.6897	-0.0256	0.7182	38s
	ML2P-VAE$_{ind}$	0.1658	0.1099	0.9939	0.0944	-0.0254	0.9976	0.6474	-0.0397	0.7579	30s

Table 4.4: Error measures for discrimination ($\boldsymbol{a}$), difficulty ($\boldsymbol{b}$), and ability ($\boldsymbol{\Theta}$) parameters from various parameter estimation methods on three different data sets. Note that in the ECPE data set, there are no true values, so MHRM estimates are accepted as true. In Sim-20, only ML2P-VAE methods are capable of estimating such high-dimensional latent traits.

more due to the number of data samples, rather than the size of the latent dimension. The affect of data size on runtime and accuracy is explored further in Section 4.2.3.1. In fact, the largest neural network we used in these experiments, used on Sim-20, only had 1,670 trainable parameters, which is very small when compared to ANN models used for image classification or natural language processing.

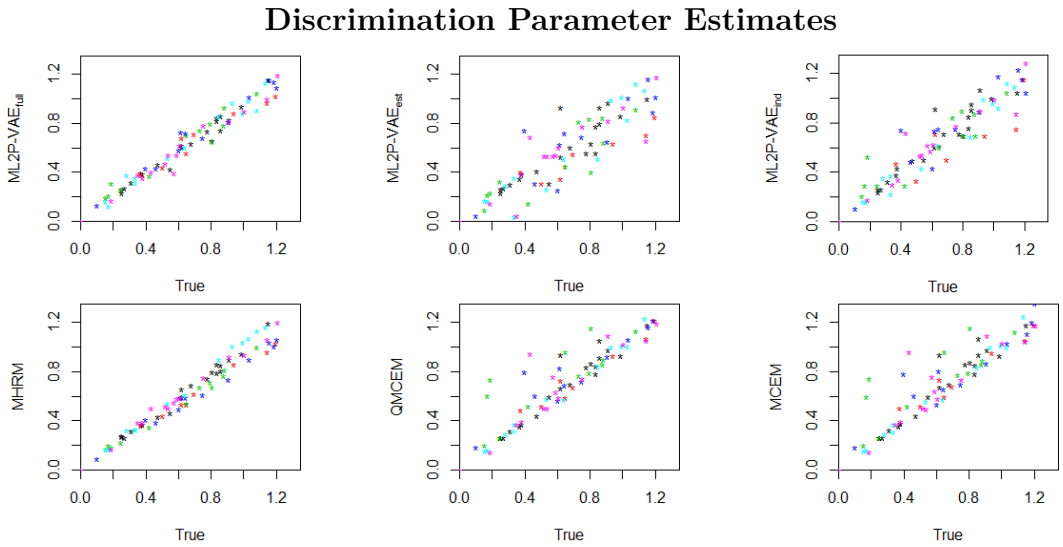

Figure 4.9: Correlation plots of discrimination parameter estimates for the Sim-6 dataset with 50 items and 6 latent traits. ML2P-VAE estimates are on the top row, and traditional method estimates are on the bottom row.

Some of the results are visualized in Figures 4.9, 4.10, 4.11, and 4.12 for Sim-6, ECPE, Sim-20, and Sim-4 respectively. Each color in the plots corresponds to a particular latent ability associated with the ability or discrimination parameter. For example, all red points correspond with estimates to θ_1 or $\boldsymbol{a}_{:1}$. Figure 4.9 shows

the correlation between the true and estimated discrimination parameters for the ML2P-VAE$_{full}$ and MHRM methods. We don't include such plots for the difficulty parameters, as all methods estimate each b_i with very high accuracy. From these figures, it appears that while MHRM obtains better results on smaller discrimination parameters, ML2P-VAE$_{full}$ has less error on larger parameters, and the estimation error seems to be independent of the magnitude of the parameter. The other two ML2P-VAE methods, ML2P-VAE$_{est}$ and ML2P-VAE$_{ind}$, do not reach the same levels of accuracy as when assuming full knowledge of the latent ability correlations.

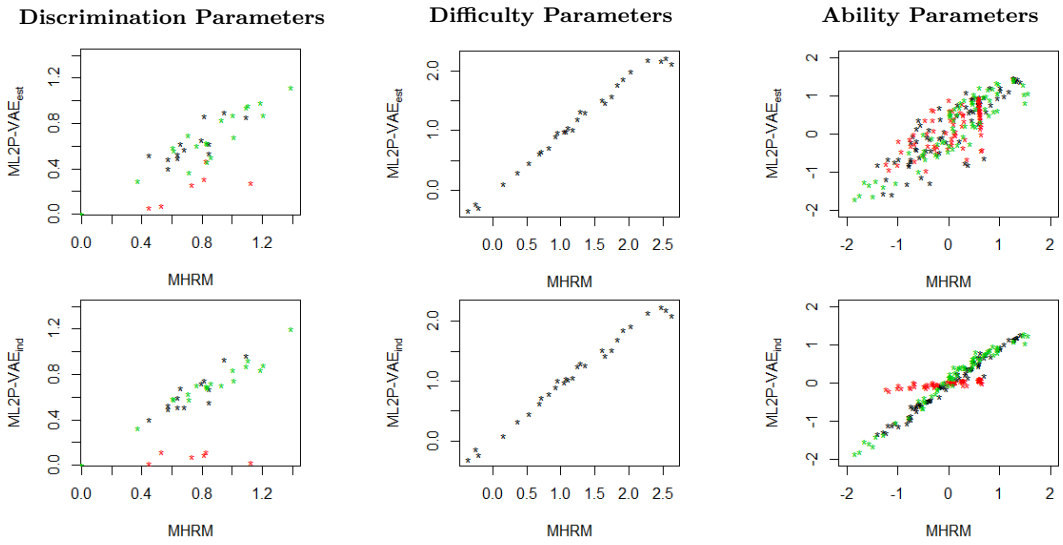

Figure 4.10: Estimates from ML2P-VAE methods plotted against "accepted" MHRM estimates from the ECPE dataset.

When examining the ECPE data, there are no "true" values of parameters so ML2P-VAE's results are directly compared with MHRM's estimates. As seen in

Table 4.4, the parameter estimates from QMCEM and MCEM are nearly identical to those of MHRM on the ECPE data. Of course, there is not a known covariance matrix between the three latent abilities, so only ML2P-VAE$_{est}$ and ML2P-VAE$_{ind}$ can be analyzed. While both methods perform similar to MHRM in difficulty parameter estimates, we can see that the two yield different results when applied to discrimination and ability parameters.

First note that while ML2P-VAE$_{ind}$ gives accurate estimations for the green and black abilities (and the discrimination parameters associated with those abilities), the red ability estimates are all very near zero for every student. This tells us that the ML2P-VAE$_{ind}$ method found that the red ability has no effect on exam performance. On the other hand, ML2P-VAE$_{est}$ captures the general trend of the MHRM ability parameters, but the estimates have much more variance. The discrimination parameter estimates also show some correlation, but each of the three abilities are on a different scale. This highlights the importance of developing better methods of estimating the latent trait correlations.

While estimating parameters for the Sim-20 dataset, the dimension of the latent traits ($\mathbb{R}^{20}$) is too large for traditional methods to handle, so only the three ML2P-VAE techniques are studied. All three of these methods estimate the difficulty parameters with high accuracy. Similar to in Sim-6, it is again observed that the ML2P-VAE$_{full}$ error seems to be independent of the size of the discrimination parameter, a promising trend. However, ML2P-VAE does not perform as well when full knowledge of the latent ability correlation matrix is unknown. At first glance, the

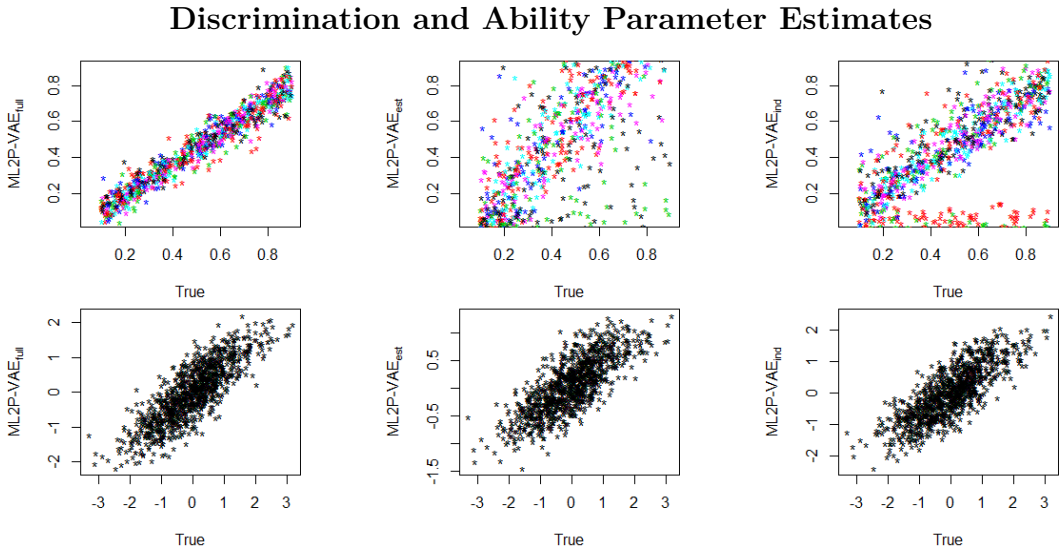

Figure 4.11: ML2P-VAE parameter estimates for Sim-20 with 200 items and 20 latent traits. The top row shows discrimination parameter correlation, and the bottom row shows ability parameter correlation.

discrimination parameter estimates for ML2P-VAE$_{est}$ seem to have no pattern. But upon closer inspection, it can be seen that the discrimination parameter estimates associated with a particular ability (a particular color) are correlated, but each ability is on a different scale.

The discrepancy between ML2P-VAE$_{full}$ and ML2P-VAE$_{est}$ can be attributed to a poorly estimated covariance matrix. For this data set, the covariance matrix obtained by the method described previously greatly overestimates every correlation between latent traits: the average signed bias $\sigma - \hat{\sigma}$ in the correlation matrix estimation computed using Equation 3.5 is -0.61, and even the closest correlation estimation has signed bias -0.26. Finding a better method to compute an approximate

correlation matrix could greatly improve ML2P-VAE$_{est}$.

The estimates for the Sim-20 dataset produced by ML2P-VAE$_{ind}$ display the same behavior observed in the ECPE dataset. Two of the abilities have discrimination parameters estimated near zero, meaning ML2P-VAE$_{ind}$ deemed these abilities to have no relation with performance on the assessment. But in contrast to the ECPE data, Sim-20 was simulated and so it is known that this is not true. Outside of this issue, the other discrimination parameters were reasonably estimated, showing clear correlation with the true values on near a 1:1 scale.

Though ML2P-VAE$_{est}$ and ML2P-VAE$_{ind}$ have trouble converging to the true discrimination parameters, they are still able to obtain quality estimates to the ability parameters. The values in Table 4.4 for $\boldsymbol{\Theta}$ in Sim-20 are comparable to those of Sim-6. The plots in Figure 4.11 show this high correlation in all three ML2P-VAE variants.

In the Sim-4 dataset, the advantages of ML2P-VAE methods are less apparent. The runtime difference is much smaller, since traditional methods do not struggle so much when integrating over a smaller latent dimension of size 4. This also affects the accuracy of parameter estimates. The latent skill estimates are better in Sim-4 than those of data set Sim-6 for all methods, but particularly the traditional methods. For latent abilities $\boldsymbol{\Theta}$ and difficulty of items $\boldsymbol{b}$, all six methods produced similar estimates, and so these correlation plots are omitted. As seen in Table 4.4, the corresponding error measures are very close, though traditional methods are slightly more accurate.

A comparison between the Sim-4 discrimination parameter estimates is shown in Figure 4.12, which clearly visualizes the values in Table 4.4. Though all ML2P-

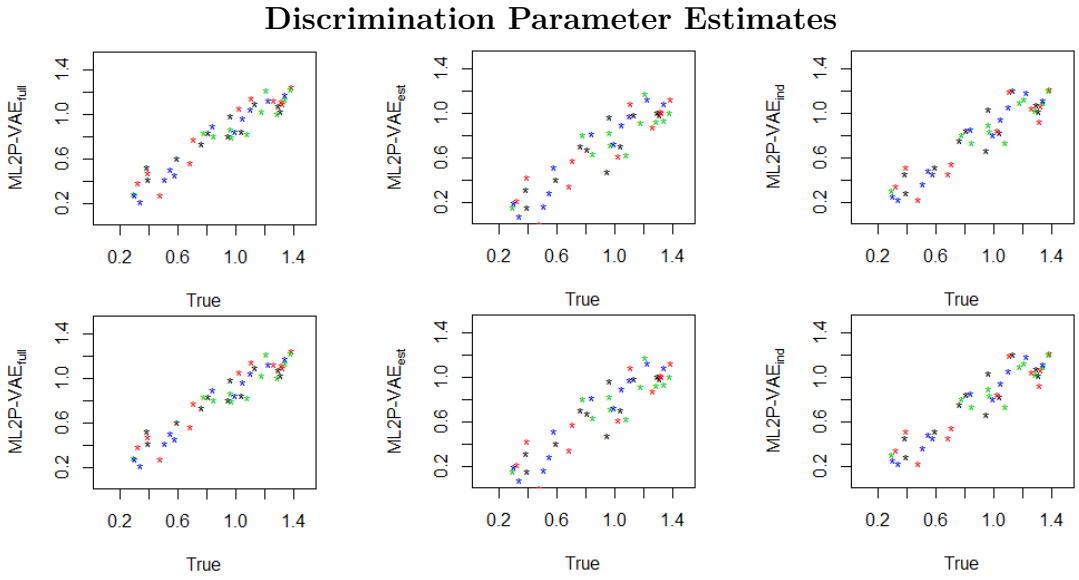

Figure 4.12: Discrimination parameter estimates for Sim-4 with 27 items and 4 latent skills. The top row shows estimates from ML2P-VAE methods, and the bottom row gives estimates yielded by traditional methods.

VAE methods produce highly correlated estimates, they also tend to underestimate the true values. This is most apparent in the plot for ML2P-VAE$_{est}$ and in the relative bias values in Table 4.4. While traditional parameter estimation results may be more desirable for the Sim-4 dataset, this demonstrates that the ML2P-VAE methods are most useful when the number of latent abilities is large.

4.2.3.1 Effect of Training Data Size

A common criticism of neural networks is that they are computationally intensive and training them with a gradient descent based algorithm (a first order method) can take a long time. They also require large amounts of data. As mentioned before,

the architecture used in this application results in a relatively small neural network.

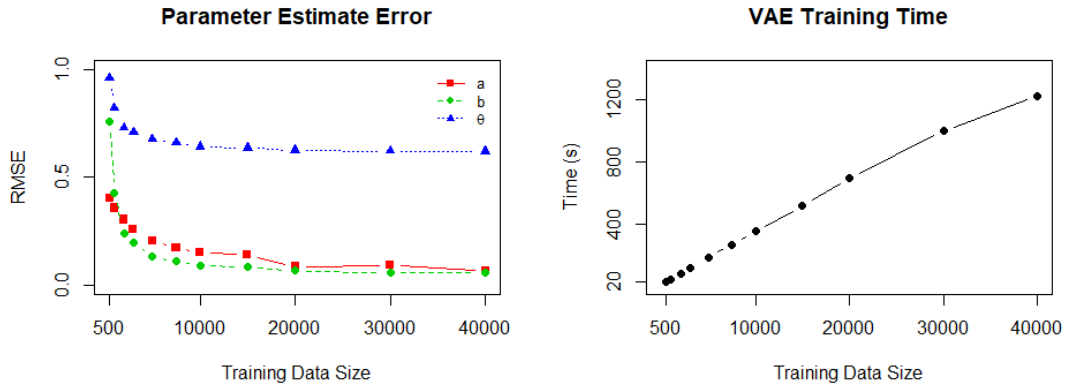

Figure 4.13: Performance of ML2P-VAE$_{full}$ on data set (iii) when trained on data sets of increasing size. The left plot gives the test RMSE after using different sizes of training data, and the right plot shows the time required to train the neural network.

The longer runtimes in Table 4.4 for Sim-20 can be attributed more to the fact that there were 50,000 data samples in the training set, rather than the large latent dimension. The left plot of Figure 4.13 displays the relation between the size of the training data and estimation accuracy. Error does not decrease very much after the number of training samples becomes greater than 20,000 – less than half of the available simulated data. The right plot of Figure 4.13 shows that training time grows linearly with the size of training data. While this may seem obvious from the usual machine learning point of view, it is not always the case in parameter estimation. The main distinction here is that the size of the gradient vector for a neural network

does not depend on the number of students N, while the gradient vector does grow with more students in traditional methods.

Both plots in Figure 4.13 demonstrate the trade-off between accuracy and speed, as well as highlighting that ML2P-VAE methods can still be viable even if the data size is not exceptionally large. This is particularly true in estimating the ability parameter $\boldsymbol{\Theta}$, whereas traditional methods are unable to estimate high-dimensional $\boldsymbol{\Theta}$. Estimating the difficulty parameters $\boldsymbol{b}$ is manageable with a smaller data set, while discrimination parameters require a large amount of training data to obtain quality estimates.

4.3 Discussion

4.3.1 Future Extensions

The work described in this chapter introduces additional paths for continued research. One important topic involves analyzing the convergence of ML2P-VAE methods. It is important to find conditions which guarantee that the estimates for the discrimination and difficulty parameters will converge to their respective true values. Based on the results shown in Table 4.4 and Figures 4.11 and 4.10, it seems likely that convergence will require full knowledge of the covariances among latent traits, since ML2P-VAE$_{full}$ performs much better than ML2P-VAE$_{est}$. In each data set, it is clear that either using an inaccurate estimated covariance matrix or simply assuming that latent traits are independent results in inaccurate parameter estimates. Another possible factor in ML2P-VAE's convergence is the sparsity of the Q-matrix.

If $q_{ik} = 1$ for all i, k, then the connections between decoder layers seen in Figure 3.1 are unmodified, and interpretation of the encoded hidden layer as estimates to ability parameters and weights/biases in the decoder as discrimination/difficulty parameter estimates may not be possible.

In real applications, it is unlikely that the exact correlations between latent abilities are available, so an approximate covariance matrix would need to be used instead. The experiments in this work imply that convergence likely relies on knowledge of an accurate covariance matrix among latent traits, thus it is important to develop better methods of estimating this covariance matrix.

It is also possible that the ML2P-VAE method can be extended to estimating the parameters in the Multidimensional Logistic 3-Parameter model [8], which introduces a guessing parameter for each item. Implementing a guessing parameter into the VAE framework is trivial. However, since many other parameter estimation methods struggle in estimating a 3-parameter model [4], "ML3P-VAE" may face the same issue. Further work to extend ML2P-VAE architecture to the Samejima graded response model [85] would allow for better analysis of student responses which are not recorded as a binary correct/incorrect. This also opens the door to other psychometric questionnaires where items are measured on a Likert scale [55].

4.3.2 Concluding Remarks

ML2P-VAE is a novel unsupervised learning technique which allows IRT parameter estimation of correlated high-dimensional latent traits. This requires a VAE

architecture capable of fitting a more general multivariate Gaussian distribution, rather than a standard normal distribution. Where other estimation methods rely on numerical integration or MCMC methods, which become infeasible for large numbers of latent abilities as described in Section 2.4, ML2P-VAE trains a neural network using a gradient descent based optimization method. Large datasets typically present a *burden* to traditional IRT parameter estimation techniques, but big data is a *blessing* to this deep learning-based approach.

While the ML2P-VAE method introduces hundreds or thousands of trainable parameters, the parameters in the decoder can be interpreted as estimates to discrimination and difficulty parameters. The individual parameters in the encoder do not represent anything concrete, but together, they learn a function which maps a student's response set to a distribution representing the student's latent ability.

All of these parameters are trained simultaneously by optimizing a single loss function. After training the neural network, the discrimination and difficulty parameter estimates are immediately available, and the ability parameter estimates are easily obtained at test time by feeding forward response sets through the encoder. Note that the estimates for $\mathbf{\Theta}_j$ are not directly trainable parameters of the neural network – the partial derivatives $\dfrac{\partial \mathcal{L}}{\partial \theta_{jk}}$ are never calculated or directly optimized.

Of course, the most accurate ML2P-VAE method, ML2P-VAE$_{full}$, makes the strongest and most restrictive assumption; that the exact correlation quantities between latent abilities is known. This may be impractical in applications, and for this reason the other ML2P-VAE methods must also be closely examined. In theory, using

a covariance matrix that is estimated from the data should yield better results than assuming all traits are independent. But if this estimated matrix is inadequate, the accuracy of parameter estimates suffers heavily. A possible way to remedy this is to adjust the weight of the KL-Divergence in the VAE loss function in Equation 2.11 by introducing a hyperparameter λ. Decreasing this hyperparameter ($\lambda < 1$) gives more emphasis on reconstructing inputs, rather than regularizing the data against an estimated distribution which may be flawed.

ML2P-VAE methods are most useful on high-dimensional data where traditional methods struggle. But even when applied to smaller data sets where traditional techniques are feasible, the results from ML2P-VAE are competitive. They are significantly faster in runtime, and yield similar error measures. When estimating difficulty parameters, the improvement gained from using traditional methods is incredibly small. Estimates for students' latent abilities are often more accurate when using ML2P-VAE methods, seen in Table 4.4. This is especially interesting, as the estimates $\mathbf{\Theta}_j$ are not updated in the iterations of a gradient descent algorithm, while the estimates to a_{ik} and b_i are. In all, these results show the versatility of ML2P-VAE methods in estimating item and ability parameters from a variety of data sets.

CHAPTER 5
TEMPORAL ANN AND KNOWLEDGE TRACING

In this chapter, we present the relevant background for Part II, the incorporation of IRT into knowledge tracing. The application of knowledge tracing extends educational measurement to a more dynamic setting. Rather than receiving all student's responses to an assessment after the exam is completed, data is gathered in real-time as the student progresses through each question. So the data is structured as a time series, and the task is to track student knowledge as more information comes in.

Because of the different format of data, this chapter first reviews popular neural networks which are equipped to deal with time series data. Though these ANN are often referred to as "recurrent" neural networks, we avoid that terminology because not all of them have a truly recurrent structure (see Section 5.3). Instead, the we classify these neural networks as "temporal neural networks," (TNN) since they relate to time-dependent data.

After providing background on various TNN in a more general setting, a literature review of previous knowledge tracing methods is provided. Specifically, we explain how TNN are adapted to better fit the knowledge tracing application, including the methods for representing student interaction data and modifications to TNN architecture.

Temporal Neural Networks

In many deep learning applications such as video processing, natural language processing, or dynamical systems, the observed data is time-dependent [31, 37, 94]. In such datasets, a single observation of n features can not represented as a vector $\boldsymbol{x}_0 \in \mathbb{R}^n$, but must take into account the T different measurements of the n features, each taken at a different timestep $1 \leq t \leq T$. As such, a data point is represented as a matrix $X_0 \in \mathbb{R}^{n \times T}$, where the t-th column of X_0 gives a snapshot of the observation at time t. For example, in the natural language application, a data point X_0 may represent a sentence or paragraph, and a column $\boldsymbol{x}_{0,t}$ of X_0 would represent a word in that sentence/paragraph.

5.1 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are the most simple adaptation of neural networks to deal with time-series data. Recall that a regular feed-forward neural network layer of size h takes an input vector $\boldsymbol{x} \in \mathbb{R}^n$ and outputs

$$\boldsymbol{y} = f(W\boldsymbol{x} + \boldsymbol{b}) \tag{5.1}$$

where $W \in \mathbb{R}^{h \times n}$ and $\boldsymbol{b} \in \mathbb{R}^h$ are trainable parameters, and f is a non-decreasing activation function [87].

In the time-dependent setting, let $\boldsymbol{x}_t$ be a column of an input X. A basic recurrent layer calculates

$$\begin{aligned} \boldsymbol{h}_t &= \tanh(W_{hh}\boldsymbol{h}_{t-1} + W_{hx}\boldsymbol{x}_t + \boldsymbol{b}_h) \\ \boldsymbol{y}_t &= \sigma(W_{hy}[\boldsymbol{x}_t, \boldsymbol{h}_t] + \boldsymbol{b}_y) \end{aligned} \tag{5.2}$$

where $W_{hh} \in \mathbb{R}^{h \times h}$, $W_{hx} \in \mathbb{R}^{h \times n}$, $\boldsymbol{b}_h \in \mathbb{R}^h$, $W_{hy} \in \mathbb{R}^{h \times (h+n)}$, and $\boldsymbol{b}_y \in \mathbb{R}^h$ are trainable parameters [32]. The notation $[\boldsymbol{x}_t, \boldsymbol{h}_t]$ refers to vector concatenation, and $\tanh(\cdot)$ and $\sigma(\cdot)$ refer to the hyperbolic tangent and sigmoid activation functions, respectively, detailed further in Appendix A.2. Note that this allows the output $\boldsymbol{y}_t$ to include information from previous timesteps. A visualization of an unfolded RNN is shown in Figure 5.1.

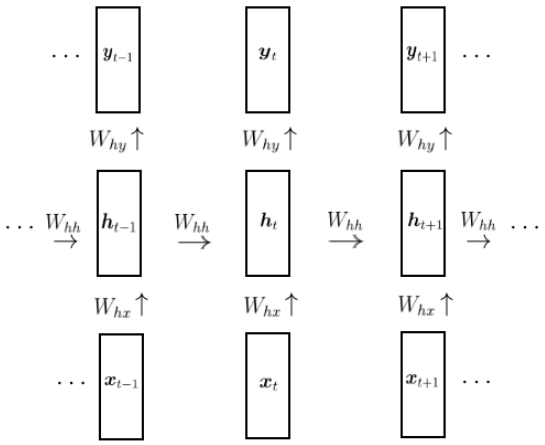

Figure 5.1: Architecture of a recurrent neural network. Note that the same weights matrices W_{hx}, W_{hh}, and W_{hy} are used in each timestep.

One issue that RNN face is the exploding or vanishing gradient problem [7], where the norm of the gradient can become very large or very small during training. This is due to the fact that partial derivatives calculated during back-propagation between hidden states at time t_1 and t_2 is found by a product of $(t_2 - t_1)$ Jacobian matrices [69]. As the difference between t_1 and t_2 increases, the corresponding partial

derivatives $\frac{\partial \boldsymbol{h}_{t_1}}{\partial \boldsymbol{h}_{t_2}}$ can exponentially grow or exponentially decay in norm.

Related to the exploding/vanishing gradient issue, RNN experience difficulty in retaining important information for multiple timesteps is difficult. For example, if an important phenomena happens to data point X_0 at time t, then that information should still influence the values of $\boldsymbol{h}_{t+10}$ and $\boldsymbol{y}_{t+10}$. But the structure described in Equation 5.2 and Figure 5.1 causes the impact of $\boldsymbol{x}_t$ and $\boldsymbol{h}_t$ to fade over time.

5.2 Long Short-Term Memory Networks

To combat the problems of RNN, Long Short-Term Memory (LSTM) networks were developed by Hochreiter and Schmidhuber [47]. This architecture introduces element-wise multiplication and addition operations along with multiple trainable weights matrices which allows for tracking long-term dependencies. An LSTM layer computes a "cell state" vector $\boldsymbol{c}_t$, in addition to the hidden layer representation $\boldsymbol{h}_t$. This cell state is updated at each timestep to "remember" important information and "forget" frivolous information.

The LSTM structure also addresses the exploding/vanishing gradient of RNN. The presence of the cell state $\boldsymbol{c}_t$ ensures that calculating derivatives of long-range dependencies do not include many matrix multiplications [47].

A single cell of an LSTM can be compared to the middle block in Figure 5.1 containing $\boldsymbol{h}_t$ of an RNN. At time t and given an input $\boldsymbol{x}_t$, previous hidden state $\boldsymbol{h}_{t-1}$, and previous cell state c_{t-1}, an LSTM cell uses four trainable weights matrices and four element-wise operations. First compute the "forget" vector $\boldsymbol{f}_t$, the "update"

vector $\boldsymbol{u}_t$, the "add" vector $\boldsymbol{a}_t$, and the "filter" vector $\boldsymbol{g}_t$.

$$\begin{aligned}
\boldsymbol{f}_t &= \sigma(W_f[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_f) \\
\boldsymbol{u}_t &= \sigma(W_u[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_u) \\
\boldsymbol{a}_t &= \tanh(W_a[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_a) \\
\boldsymbol{g}_t &= \sigma(W_g[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_g)
\end{aligned} \tag{5.3}$$

The first three vectors in Equation 5.3 are used to perform element-wise operations on $\boldsymbol{c}_{t-1}$ to produce the next cell state $\boldsymbol{c}_t$, and $\boldsymbol{g}_t$ is used in updating $\boldsymbol{h}_t$. Notice that the sigmoid activation function $\sigma(\cdot)$ maps small inputs to near 0 and large inputs to near 1, while the hyperbolic tangent activation function $\tanh(\cdot)$ maps small inputs to near -1 and large inputs to near 1.

Using $\boldsymbol{f}_t$, unimportant aspects (elements in $\boldsymbol{f}_t$ near zero) of $\boldsymbol{c}_{t-1}$ are forgotten:

$$\tilde{\boldsymbol{c}}_{t-1} = \boldsymbol{c}_{t-1} \times \boldsymbol{f}_t \tag{5.4}$$

where $\times$ is element-wise multiplication. Next, $\boldsymbol{u}_t$ decides what information to update (elements in $\boldsymbol{u}_t$ near one), and $\boldsymbol{a}_t$ gives the value (an increase or decrease) of the information to be updated:

$$\boldsymbol{c}_t = \tilde{\boldsymbol{c}}_{t-1} + (\boldsymbol{u}_t \times \boldsymbol{a}_t) \tag{5.5}$$

where $+$ and $\times$ are element-wise addition and multiplication, respectively. Lastly, compute the next hidden state using $\boldsymbol{g}_t$, which filters the information to be passed to the next network layer and next timestep:

$$\boldsymbol{h}_t = \tanh(\boldsymbol{c}_t) \times \boldsymbol{g}_t \tag{5.6}$$

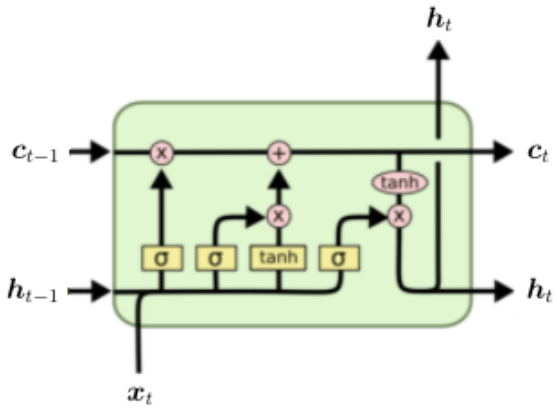

Figure 5.2: Architecture of a single LSTM cell [66]. Trainable matrix multiplication followed by an activation function are in yellow boxes, and element-wise operations without learned parameters are in red ovals.

The architecture of a single LSTM cell is visualized in Figure 5.2. The forget vector acts as a gate which allows/disallows past information to persist over time, while the update and add vector grabs the data from the current input which is worth updating and remembering.

5.3 Transformers and Attention

Though LSTM networks presented a significant breakthrough in natural language processing (NLP), they have been quickly surpassed in the application of language modeling by attention-based methods. These models forgo the recurrent structure of information flow seen in RNN and LSTM for feed-forward layers and similarity scores between time steps. For example, calculating a similarity score between each pair of words in a sentence can help extract deeper context in language models such

as transformers [94].

While transformers are large neural networks with many components and parameters, they lean heavily on a simple attention mechanism. Given an n-dimensional feature vector $\boldsymbol{x}_t$ at each timestep $1 \leq t \leq T$, define three trainable matrices $W_q, W_k, W_v \in \mathbb{R}^{n \times d}$. These are used to obtain a *query*, *key*, and *value* vectors $\boldsymbol{q}_t$, $\boldsymbol{k}_t$, and $\boldsymbol{v}_t$ for each timestep.

The query and key vectors are representations of the observations $\boldsymbol{x}_t$ which are used to quantify the relationship between observations $\boldsymbol{x}_{t_1}$ and $\boldsymbol{x}_{t_2}$, with the distinction between them being that a query typically refers to the current timestep. In other words, if event t just occurred, we may be interested in comparing $\boldsymbol{q}_t$ to a previous observation $\boldsymbol{k}_{t-1}$. The value vector $\boldsymbol{v}_t$ is a representation of $\boldsymbol{x}_t$ which holds deeper, more contextual information of the observation. Note that $\boldsymbol{q}_t$, $\boldsymbol{k}_t$, and $\boldsymbol{v}_t$ can be stacked into matrices $Q, K, V \in \mathbb{R}^{T \times d}$.

The queries and keys are used first, to calculate the correlation between observation t and all other timesteps:

$$\boldsymbol{c}_t = \text{softmax}\left(\frac{K\boldsymbol{q}_t}{\sqrt{d}}\right) \in \mathbb{R}^T \tag{5.7}$$

Notice that the matrix multiplication $K\boldsymbol{q}_t$ in Equation 5.7 is simply T individual dot-product computations. So the i-th entry of $\boldsymbol{c}_t$ gives the similarity between the input at time t and the input at time i. The softmax function $\text{softmax}(z_i) = \dfrac{e^{z_i}}{\sum_{j=1}^{T} e^{z_j}}$ rescales the dot product calculations so that the sum of the entries of $\boldsymbol{c}_t$ is equal to 1.

In applications where an input $\boldsymbol{x}_{t_1}$ is not allowed to see information of future inputs $\boldsymbol{x}_{t_2}$, $t_1 < t_2$, the corresponding entries of $K\boldsymbol{q}_{t_1}$ are masked to be $-\infty$. This

causes the entries $c_{ti} = 0$ when $i > t$. It is also worth pointing out that this correlation score is not symmetric – the dot product between two queries and keys of the timesteps $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t'}$ is not equal: $\boldsymbol{k}_{t'}^\top \boldsymbol{q}_t \neq \boldsymbol{k}_t^\top \boldsymbol{q}_{t'}$. To understand why this is desirable, consider the NLP application and the sentence "The dog is happy." Correlation between a query for "The" and a key for "dog" should be different from the reverse. When querying "The," we wish to know what "The" is referring to (the dog). But when querying "dog," we wish to know something about the dog (it is happy) – the article "The" is unimportant in this sense.

Next, the attention is calculated as

$$\boldsymbol{a}_t = V\boldsymbol{c}_t \in \mathbb{R}^d \tag{5.8}$$

The attention vector $\boldsymbol{a}_t$ is a weighted sum of the value vectors of each other timestep, weighted by the correlation scores in Equation 5.7. The attention calculation can also be written more generally for all timesteps at once:

$$A = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right) V \in \mathbb{R}^{T\times d} \tag{5.9}$$

In attention networks, the attention vectors $\boldsymbol{a}_t$ are individually sent through feed-forward layers. For example, transformers calculate attention and then use three feed-forward layers in a single "block" [94]. These blocks are stacked on top of each other up to six times to obtain deeper and deeper contextualization of the input sequence [26]. Eventually, this contextualization is plugged into a final prediction layer, depending on the application.

It should be pointed out that transformers and attention networks were de-

veloped specifically for the NLP application. So in this case, a sequence of T inputs represents a sentence of length $\leq T$ and an input vector $\boldsymbol{x}_t$ is an n-dimensional learned representation of a single word [61]. Then the correlation score in Equation 5.7 is quantifying the relationship between pairs of words in the sentence.

Knowledge Tracing Overview

Knowledge Tracing (KT) is a task introduced by Corbett and Anderson in 1995 [23]. Their goal was to model the changing knowledge state of students as they progress through an online intelligent tutoring program. This tutoring system helps students practice writing computer programs by testing them on various rules, such as correct use of in-built functions, and providing feedback on their mistakes. The model tracks each student's knowledge as being in either a learned or unlearned state for each rule. After each interaction, there is a probability $P(T)$ that a student makes the transition from the unlearned state to the learned state.

The probability that a student has learned a particular rule at timestep t is given by

$$P(L_t) = P(L_{t-1}|\text{evidence}) + (1 - P(L_{t-1}|\text{evidence})) \cdot P(T), \tag{5.10}$$

where "evidence" refers to whether or not previous interactions were correct or not [23]. The probability of student performing the t-th task correctly is

$$P(c_t = 1) = P(L_t) \cdot (1 - P(s)) + (1 - P(L_t)) \cdot P(g) \tag{5.11}$$

where $P(s)$ is the probability that the student slips (makes a silly mistake) and $P(g)$ is the probability that the student randomly guesses correctly. The parameters $P(T)$,

$P(g)$, and $P(s)$ are estimated using hidden Markov model, along with an initial value $P(L_0)$. Then Equation 5.10 is updated via Bayes' rule.

In recent years, Bayesian Knowledge Tracing (BKT) has been overcome by deep learning methods. The popularity of deep neural networks has brought black-box models that yield high accuracy. Many of these methods, detailed in Section 5.4, do not provide a concrete measure of student ability over time. Instead, the only way to track student knowledge is through the predicted probability of them answering questions correctly at a given timestep.

In Chapter 6, new methods using neural networks are presented which produce comparable predictive power to deep learning methods, while providing explainable models with links to Item Response Theory.

5.4 Knowledge Tracing Literature Review

In the modern knowledge tracing application, data is provided as a sequence of student interactions $x_t = (q_t, c_t)$, $0 \leq t \leq L$. L is a hyperparameter denoting the maximum length of the sequence – since the number of interactions for each student is different, response sequences shorter than L are padded with null interactions, and response sequences of length longer than L are wrapped into multiple sequences. For example, if $L = 128$ and a particular student answers 161 questions, then this student's interactions will be split into two separate sequences of length 81 and 80.

The tag q_t indexes a particular question (item) in the available question bank, and $c_t \in \{0, 1\}$ indicates whether the question was answered incorrectly or correctly.

So for a learning system with n available questions, there are $2n$ distinct interactions for the tuple x_t. The knowledge tracing task is to predict c_{t+1} given all previous interactions. Mathematically, the quantity of interest is the probability

$$P(c_{t+1} = 1|(q_0, c_0), (q_1, c_1), \ldots, (q_t, c_t), (q_{t+1}, ?)). \tag{5.12}$$

Most neural networks optimize the predicted probability in Equation 5.12 by minimizing the cross-entropy loss function, as described in Equation 2.2.

5.4.1 Deep Knowledge Tracing

In 2015, the first use of neural networks for knowledge tracing was introduced by Piech et al. [73]. Deep Knowledge Tracing (DKT) utilizes recurrent neural networks (RNN) and Long-Short Term Memory (LSTM) neural networks to predict a student's success on future questions, given a sequence of previous interactions.

Similar to natural language processing, tokens (student interactions) need to be represented as a d-dimensional vector. DKT does this by one-hot encoding the interactions in the input layer to have shape $(2n + 1, L)$, and linearly mapping to a hidden layer of shape (d, L). Each interaction in the sequence is treated independently in this layer. The input layer shape is $2n + 1$ for each of the possible $2n$ interactions, along with space for an additional padding token representing a null interaction (for response sequences of length $< L$.

The architecture of DKT is as follows: The one-hot encoding input layer is followed by a d-dimensional embedding. Then, an LSTM layer of size d, and a feed-forward output layer with n nodes. The final layer uses a sigmoid activation

function, and the output at each node represents the probability of answering the corresponding item correctly at the given timestep. To calculate loss, only the item tag for the next interaction and corresponding output node is used in the cross-entropy loss calculation. Output nodes corresponding with items that don't match the interaction sequence are masked out.

5.4.2 Dynamic Key-Value Memory Networks

A more sophisticated neural network approach to knowledge tracing was introduced by Zhang et al. with Dynamic Key-Value Memory Networks (DKVMN) [101]. They modify a memory-augmented neural network (MANN) in order to fit into the knowledge tracing framework. A MANN is a temporal neural network which uses a value matrix M^v stored in memory for each student. The entries in M^v are updated in each timestep with element-wise addition and multiplication, similar to an LSTM in Section 5.2. The output is a probability dependent on the previous value of M^v in timestep $t-1$, as well as the current neural network input in timestep t.

In DKVMN, there is some added interpretability by requiring the number of columns of M^v to be equal to the number of knowledge concepts K. In this way, the columns of M^v offer an h-dimensional representation of the student's skill. DKVMN splits the computations into two parts: *read* from M^v to make a prediction of answering an item correctly, and *write* to M^v to update its information after observing a true outcome. The predictive part inputs only an exercise tag q_t without the true response c_t. The question tag is linearly embedded into a vector $\boldsymbol{k}_t$, so that

$\boldsymbol{k}_t$ is a representation of question q_t, and is then multiplied by a learned matrix M^k and softmaxed:

$$\boldsymbol{w}_t = \text{softmax}\left(M^k \boldsymbol{k}_t\right) \tag{5.13}$$

This creates a vector $\boldsymbol{w}_t$, where entry i in $\boldsymbol{w}_t$ represents the correlation weight between the question q_t and memory slot i. This process of taking the dot product between an item embedding and a trainable matrix then softmaxing is similar to the concept of attention (described in Section 5.3), used in popular NLP techniques such as transformers [94].

Next, *read* from the value matrix by computing

$$\boldsymbol{r}_t = \sum_{i=1}^{K} \boldsymbol{w}_t(i) M_t^v(i). \tag{5.14}$$

Note that $\boldsymbol{r}_t$ is simply a weighted sum of the columns of M^v and can be treated as a summary of the student's predicted mastery level of exercise q_t. Next, the item embedding $\boldsymbol{k}_t$ is appended to the read content $\boldsymbol{r}_t$ and fed forward through two linear layers. The first uses a tanh activation function, and the output p_t produces a single node with sigmoidal activation. In this way, the single value p_t represents the probability that the student will answer item q_t correctly at timestep t.

The second part of DKVMN is to *write* new values into M^v based on the true response of students. Different from the prediction phase, the full tuple (q_t, c_t) is embedded into a vector $\boldsymbol{v}_t$. The manner in which M^v is updated is actually similar to that of an LSTM, allowing for "remembering" and "forgetting". $\boldsymbol{v}_t$ is multiplied by two trainable matrices to produce an "erase" vector $\boldsymbol{e}_t$ and an "add vector" $\boldsymbol{a}_t$.

The erase vector has a sigmoidal activation function, so that values near zero do not get erased much at all, and values near one are mostly erased. The add vector uses a tanh activation function, so memory slots in M^v can either be increased or decreased. Finally, the columns of the memory matrix are updated via

$$M_t^v(i) = (M_{t-1}^v(i)[\mathbf{1} - \boldsymbol{w}_t(i)\boldsymbol{e}_t]) + \boldsymbol{w}_t(i)\boldsymbol{a}_t \tag{5.15}$$

Note that the correlation weights $\boldsymbol{w}_t$ computed in the predictive step are again used to determine *how much* of memory slot i should be updated.

DKVMN's use of a matrix stored in memory allows for longer range dependencies than RNN or LSTM. There is also a bit of interpretability in this method, since a single column of the memory matrix M_t^v gives an h-dimensional representation of a single skill for the student at time t. However, it cannot be determined *which* skill the column represents, as they could be permuted in any order. Additionally, if a student answers each available item, then stacking each weights vector $\boldsymbol{w}_t$ into a matrix $W = \{\boldsymbol{w}_t\}_{t=1}^L$ should result in a matrix similar to the item-skill association Q-matrix. But again, the columns of this "learned Q-matrix" W are in no particular order, and can be difficult to interpret.

5.4.2.1 Deep-IRT

Deep-IRT, proposed by Chun-Kit Yeung [99], modifies the DKVMN architecture to allow a connection with Item Response Theory. Specifically, two separate feed forward layers are inserted representing θ_{tk}, a student's k-th ability at time t and concept difficulty β_k. Then the output probability is not another linear layer (as in

DKVMN), but is instead a function of θ_{tk} and β_k:

$$p_t = \frac{1}{1 + \exp\left(\beta_k - 3 \cdot \theta_{tk}\right)} \tag{5.16}$$

These modifications provide a link to the Rasch model in Equation 2.15. The multiplication by 3 is for practical reasons to re-scale θ_{tk}. However, note that in Equation 5.16, the difficulty parameter is on the *concept* level, and not the *item* level like the Rasch model (and other IRT models). Though Deep-IRT doesn't seek to directly approximate the Rasch model, the modifications to DKVMN still adds significant interpretability to the deep neural network.

5.4.3 Self-Attentive Knowledge Tracing

In the field of natural language processing (NLP), the most state-of-the-art methods utilize the self-attention mechanism [94], which relies on calculating the correlation between pairs of words in a sentence. Popular models such as BERT [29] and GPT-3 [12] are both transformer-based neural networks for NLP which heavily depend on attention. Self-Attentive Knowledge Tracing (SAKT) adapts this concept for the knowledge tracing task [68].

Similar to other deep learning methods, at timestep t, SAKT first embeds each interaction (q_i, c_i), $i < t$ into a learned d-dimensional vector $\boldsymbol{m}_i$. Additionally, like DKVMN, the current question q_t without the response is also embedded into a d-dimensional vector $\boldsymbol{e}_t$.

The exercise embedding $\boldsymbol{e}_t$ is multiplied by a weights matrix to obtain a *query*

vector $\boldsymbol{q}_t = W_q \boldsymbol{e}_t$. The interaction embedding $\boldsymbol{m}_i$ is used to create two vectors: a *key* vector $\boldsymbol{k}_i = W_k \boldsymbol{m}_i$ and a *value* vector $\boldsymbol{v}_i = W_v \boldsymbol{m}_i$.

The general idea is that $\boldsymbol{k}_i$ serves as the identifier of a past interaction, and $\boldsymbol{q}_t$ serves as an identifier for the current exercise. If the two exercises are similar in content, then the dot product between these two vectors should be large. The value vector $\boldsymbol{v}_i$ holds more abstract information about the corresponding interaction. The keys and values are organized into matrices K and V. We calculate the attention

$$\boldsymbol{a}_t = \text{softmax}\left(\frac{K\boldsymbol{q}_t}{\sqrt{d}}\right)V \tag{5.17}$$

The value $\frac{K\boldsymbol{q}_t}{\sqrt{d}}$ yields a vector where the i-th entry is the dot product between the current exercise query $\boldsymbol{q}_t$ and an interaction key $\boldsymbol{k}_i$. This is scaled by dimension and softmaxed, resulting in a weighted sum of the value vectors $\boldsymbol{v}_i$.

The attention value $\boldsymbol{a}_t$ is sent through a few feed-forward layers, resulting in a vector $\boldsymbol{f}_t = \text{FFN}(\boldsymbol{a}_t)$. The output layer is $p_t = \sigma(\boldsymbol{f}_t W + \boldsymbol{b})$, the probability that the student will answer the current exercise q_t correctly, where W and b are trainable parameters.

5.4.4 Performance Factors Analysis

An earlier approach to knowledge tracing was proposed by Pavlik et al. in 2009 with Performance Factors Analysis (PFA) [71]. The general idea is that a student's learning at a given timestep is a function of the student's past interactions with items related to various knowledge concepts. Specifically, the logit of student j answering item i correctly is a linear combination of concept difficulty, previous successes, and

previous failures:

$$\begin{aligned} \log \frac{p}{1-p} &= \sum_{k \in K_i} (\beta_k + \gamma_k s_{jk} + \rho_k f_{jk}) \\ p(c_i = 1 | k \in \mathrm{K}_i, \boldsymbol{s}_j, \boldsymbol{f}_j) &= \sigma \left(\sum_{k \in K_i} (\beta_k + \gamma_k s_{jk} + \rho_k f_{jk} \right) \end{aligned} \tag{5.18}$$

In Equation 5.18, K_i is a set indicating which knowledge concepts are required for item i, the trainable parameter β_k represents concept k's difficulty, and γ_k and ρ_k serve as trainable weights. s_{jk} and f_{jk} track the prior successes and failures, respectively, of student j on concept k. At timestep t, we can write s_{jk} and f_{jk} as

$$\begin{aligned} s_{jk} &= \sum_{i<t} \chi_{(c_i=1)} \cdot \chi_{(k \in \mathrm{K}_i)} \\ f_{jk} &= \sum_{i<t} \chi_{(c_i=0)} \cdot \chi_{(k \in \mathrm{K}_i)} \end{aligned} \tag{5.19}$$

where χ is the indicator function. For example, $\chi_{(k \in \mathrm{K}_i)}$ indicates whether the previous item q_i, $i < t$, required knowledge concept k or not.

The parameters β_k, γ_k, and ρ_k are learned so that they maximize the log-likelihood of the given dataset. This is a well-studied problem, as the form of Equation 5.18 is essentially a logistic regression. Note that similar to Deep-IRT, PFA focuses on the concept-level, rather than item-level, parameters.

5.4.5 Deep Performance Factors Analysis

Recent work has related PFA to the self-attention mechanism used in SAKT described in Section 5.4.3. Pu et al. [75] developed Deep Performance Factors Analysis (DPFA) and a new characterization of the weight parameters γ_k and ρ_k, using learned item embeddings $\boldsymbol{e}_i$ for each question q_i.

For the current question q_{t+1}, the attention between previous exercises is $A_{i,t+1} = \boldsymbol{e}_i^\top \boldsymbol{e}_{t+1}$, for $i \leq t$. Note that this incorporates ideas from SAKT, but the attention is symmetric. More recent interactions are taken into account by calculating $d_{i,t+1} = -a(t-i+1)+b$, where a and b are trainable parameters. Then the relevance of a past item depends on the dot product similarity and how long ago the interaction took place:

$$w_{t+1} = \text{softmax}(A_{:,t+1} + d_{:,t+1}) \tag{5.20}$$

The mastery of knowledge concepts after a student completes interaction (q_i, c_i) is given as $\boldsymbol{v}_i = [v_i^0, v_i^1] \in \mathbb{R}^2$. The numbers v_i^0 and v_i^1 represent the expected mastery of the skills for item i if the item is answered incorrectly or correctly, respectively. DPFA gives the probability of a student answering item q_{t+1} correctly as

$$p_{t+1} = \sigma\left(\beta_{t+1} + \sum_{i \leq t} \left(\chi_{(c_i=0)} \cdot w_i v_i^0 + \chi_{(c_i=1)} \cdot w_i v_i^1\right)\right) \tag{5.21}$$

where β_{t+1} corresponds to the difficulty of the current item and $\sigma(\cdot)$ is the sigmoid function. In comparison with regular PFA in Equation 5.18, DPFA substitutes the terms $\chi_{(c_i=0)} \cdot w_i v_i^0$ for $\rho_k f_k$ and substitutes $\chi_{(c_i=1)} \cdot w_i v_i^1$ for $\gamma_k s_k$.

CHAPTER 6
DEEP, INTERPRETABLE KNOWLEDGE TRACING METHODS

The deep neural network approaches described in Chapter 5 (including DKT, DKVMN, and SAKT) to the knowledge tracing problem have resulted in very high predictive power. Given a sequence of previous student interactions and a current question, they are capable of outputting the probability that the current question will be answered correctly with high accuracy. For the most part, this is the only metric produced by these models to measure student learning – are these deep methods actually *tracing* student knowledge? There already exists a theoretical framework for computing the probability of a correct response: Item Response Theory, introduced in Section 2.3.

In this section, we introduce an interpretable modification to deep knowledge tracing methods, originally proposed in "Incorporating Item Response Theory into Knowledge Tracing" published in the proceedings of AIED 2021 [22]. Specifically, we link Item Response Theory models into the structure of knowledge tracing neural architecture. Besides the theoretical advantages this gives to a knowledge tracing model, it is also very helpful in practice. First, it provides an accessible and explicit representation of student knowledge at each timestep. This is an upgrade from other deep knowledge tracing methods, where the only meaningful values produced is p_{t+1}, the probability of answering the next question correctly (Equation 5.12). The proposed modification also functions as a parameter estimation technique, quantifying the difficulty and discrimination power of items.

The connection between knowledge tracing and IRT has been explored before via the Deep-IRT method. The IRT-inspired knowledge tracing methods presented here differ from Deep-IRT in a few ways. First, Deep-IRT is tightly coupled with DKVMN, while the proposed method is readily applicable to a variety of deep knowledge tracing models. Our approach also allows for items to be associated with multiple skills, and directly emulates the ML2P model in Equation 2.18 or the Rasch model in Equation 2.15 by producing estimates to discrimination and difficulty parameters. The focus here involves item-level parameters, rather than the concept-level parameters considered in Deep-IRT. The implementation details we use are completely different from that of Deep-IRT. Rather than adding separate networks for each parameter, we modify the output layer using information from the item-skill association Q-matrix, similar to the methodology of ML2P-VAE in Section 3.1.

In this chapter, a trade-off is presented between predictive power and interpretability, but the proposed method remains competitive with other deep learning methods. While sacrificing a small amount of accuracy (measured by AUC), IRT-inspired knowledge tracing provides an explicit representation of student knowledge $\boldsymbol{\Theta}$ at each timestep. This representation of student knowledge is an upgrade from other deep knowledge tracing methods, which approximate skill mastery after the fact by averaging the probability of correctly answering all items associated with a particular skill. Additionally, parameters of the proposed modified neural network can be interpreted as approximations to the item parameters a_{ik} and b_i in Equation 2.18. In this sense, our proposed models function as both a knowledge tracing model

and a parameter estimation method.

6.1 Incorporating IRT into Knowledge Tracing

Given a tutoring system with n available items, K skills under assessment, and the skill association of each item given as a binary matrix $Q \in \{0,1\}^{n \times K}$ [25], each of the possible $2n$ student interactions (q_t, c_t) is represented as a learned d-dimensional vector $\boldsymbol{x}_t \in \mathbb{R}^d$. This can be done by multiplying a one-hot encoding of the $2n+1$ interactions (including a null/padding interaction) by a trainable $(2n+1) \times d$ matrix.

Each student's response sequence includes $\boldsymbol{x}_0$, a null interaction to indicate the start of their interactions. A hyperparameter L is chosen indicating the maximum length of a student's response sequence. Interaction sequences shorter than L are padded, and interaction sequences longer than L are split into multiple sequences. A student's sequence of embedded interactions $\{\boldsymbol{x}_t\}_{t=0}^{L}$ is fed through a temporal neural network (TNN), such as an LSTM (similar to DKT [73]) or an attention-based model (similar to SAKT [68]). This outputs an h-dimensional vector $\boldsymbol{v}_t$ for each interaction $\boldsymbol{x}_t$ in the input sequence.

$$\boldsymbol{v}_t = \mathrm{TNN}(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}, \ldots, \boldsymbol{x}_0), \quad \boldsymbol{v}_t \in \mathbb{R}^h \tag{6.1}$$

Next, each $\boldsymbol{v}_t$ is sent through a linear layer feed-forward network with output size K (the number of latent concepts), yielding the skill vector $\boldsymbol{s}_t$.

$$\boldsymbol{s}_t = W_s \boldsymbol{v}_t + \boldsymbol{y}, \quad W_s \in \mathbb{R}^{K \times h}, \boldsymbol{y} \in \mathbb{R}^K \tag{6.2}$$

The weights matrix W_s and bias vector $\boldsymbol{y}$ are trainable parameters. Each node in

this "skill layer" represents a knowledge concept.

Finally, the output layer of the model has n nodes and a sigmoid activation function $\sigma(\cdot)$, with each node representing the probability of the student answering that item correctly.

$$\boldsymbol{p}_t = \sigma(W_p \boldsymbol{s}_t + \boldsymbol{z}) = \frac{1}{1 + \exp\left(-W_p \boldsymbol{s}_t - \boldsymbol{z}\right)}, \quad W_p \in \mathbb{R}^{n \times K}, \boldsymbol{z} \in \mathbb{R}^n. \tag{6.3}$$

W_p and $\boldsymbol{z}$ are trainable and importantly, W_p is modified so that the nonzero values of W_p are determined by the Q-matrix [24, 39]. If item i does not require skill k, then the weight between the corresponding nodes is fixed to be zero. In this way, we write

$$W_p \leftarrow W_p \odot Q, \tag{6.4}$$

where $\odot$ is element-wise multiplication of matrices. Then the probability that the student will answer question i correctly at timestep t is given by

$$p_{ti} = \frac{1}{1 + \exp\left(-\sum_{k=1}^{K} w_{ik} q_{ik} s_{tk} - z_i\right)} \tag{6.5}$$

where w_{ik}, q_{ik}, s_{tk}, and z_i are entries in W_p, Q, $\boldsymbol{s}_t$, and $\boldsymbol{z}$, respectively.

This constraint allows for interpretation of the final neural network layers as an approximate ML2P model: note the similarity between Equation 6.5 and Equation 2.18. A visualization of this proposed neural network architecture is seen in Figure 6.1.

The weights between the skill and output layer ($w_{ik} q_{ik}$) serve as estimates to the discrimination parameters a_{ik}, and the bias parameters in the output layer z_i are

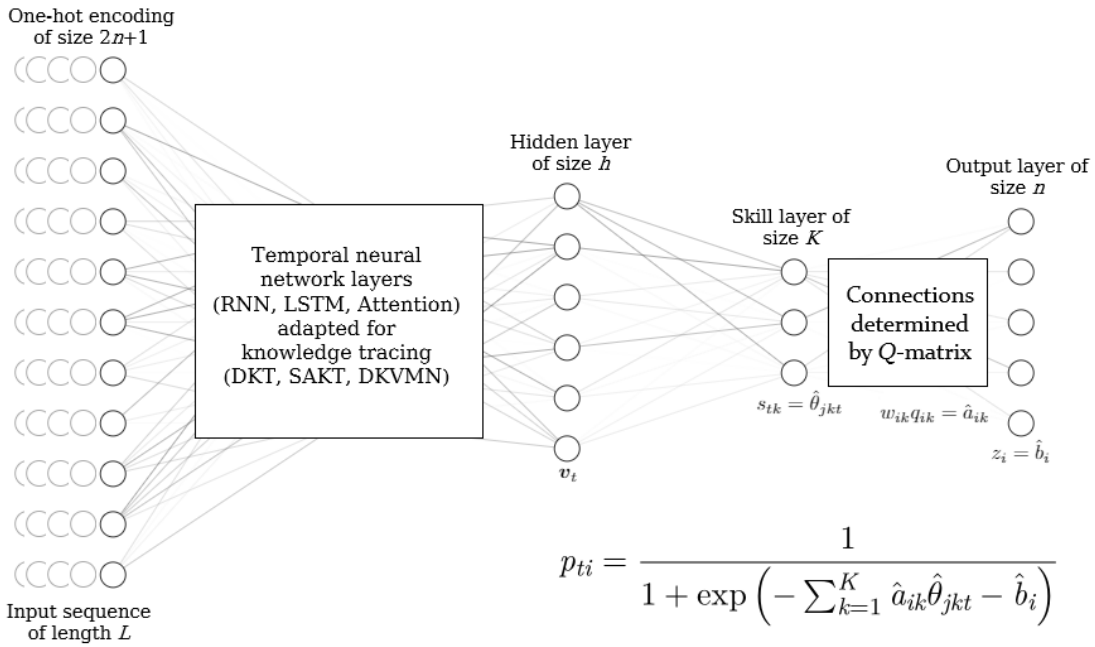

Figure 6.1: Visualization of integrating IRT into a knowledge tracing model with $L = 4$, $n = 5$, and $K = 3$.

estimates to difficulty parameters b_i. The student's k-th latent ability θ_k is estimated at timestep t via s_{tk}. This presents a clear analogue with ML2P-VAE, but the method described in this section makes no assumption about the prior distribution of latent traits $\mathbf{\Theta}$ – there is no KL-Divergence term in the knowledge tracing loss function. In this way, IRT-inspired knowledge tracing is more similar to the modified autoencoder proposed by Guo et al. [39] and described in Section 4.2.2.

CHAPTER 7
KNOWLEDGE TRACING RESULTS AND DISCUSSION

7.1 Description of Datasets

We use four publicly available datasets, three of which are standard in the knowledge tracing literature. Two of the four datasets are simulated according to IRT models to demonstrate the capability of our IRT-inspired knowledge tracing methods to learn the IRT model parameters. We also include two real-world datasets common in knowledge tracing literature. A summary of each dataset is given in Table 7.1.

Synth5

This dataset [1] was generated by Piech et al. [73] for experiments with DKT. There are 50 items covering 5 latent concepts. Each item requires exactly one concept, and responses are generated according to the Rasch model [57] with guessing:

$$P(u_{ij} = 1|\Theta_j; b_i) = c + \frac{1-c}{1+e^{b_i-\theta_{jk}}} \tag{7.1}$$

The guessing parameter c is fixed at 0.25. Note that in Equation 7.1, all items are simple items – only a single skill k is referenced when answering item i. Responses to each of the 50 items are simulated for 4,000 students.

[1] https://github.com/chrispiech/DeepKnowledgeTracing/tree/master/data/synthetic

Sim200

Sim200 [2] differs from Synth5 in a few important ways. First, there are more items (200) and more latent skills (20). Second, the Q-matrix is more dense – items require multiple skills in order for students to answer correctly. Each entry in the Q-matrix was sampled from Bern(0.2). Lastly, Sim200 generates responses according to the ML2P model in Equation 2.18, as opposed to the Rasch model. The item parameters were taken from a random uniform distribution; the difficulty parameters from $b_i \in [-3, 3]$ and the nonzero discrimination parameters from $a_{ik} \in [0.1, 0.9]$. This dataset is very similar to the Sim-20 dataset described in Section 4.1, but the latent abilities $\boldsymbol{\Theta}$ were generated according to a standard normal Gaussian distribution.

Statics2011

Statics [3] is a real-world dataset with responses from 316 students enrolled in a college engineering course. After formatting the data (removing a student's multiple attempts on the same item), the dataset includes 987 unique items and 61 latent concepts. Students answered varying amounts of questions, with a total of 135,338 distinct interactions.

---

[2] https://github.com/converseg/irt_data_repo/tree/master/Sim200

[3] https://pslcdatashop.web.cmu.edu/DatasetInfo?datasetId=507

Dataset	Items	Skills	Students	Interactions
Synth5	50	5	4,000	20K
Sim200	200	20	50,000	10M
Statics2011	987	61	316	135K
Assist2017	4,117	102	1,709	392K

Table 7.1: Summary of knowledge tracing datasets.

Assist2017

The ASSISTments 2017 dataset [4] contains real-world interactions from 1,709 students recorded on the ASSISTments online tutoring system. There is a large number of distinct items (4,117), and 102 latent concepts. Some items are tagged with the concept "no skill" – we treat this tag as a distinct latent concept, otherwise all interactions involving such items would need to be thrown out.

Experiment Details

In the two simulated datasets (Synth5 and Sim200), all students answer the same set of questions and thus all have the same length of response sequences (50 and 200, respectively). For these simulated datasets, the order of responses is shuffled randomly for each student – the importance of this permutation is discussed in Section 7.2.1. On Statics2011 and Assist2017, the maximum sequence length is set at $L = 128$, and a student whose response sequences are longer/shorter than 128 interactions have their response sequences wrapped/padded. The rest of the hyperparameters are described in Table 7.2. The hyperparameters for DKT, SAKT, and DKVMN follow

[4] https://sites.google.com/view/assistmentsdatamining

Parameter	Synth5	Sim200	Statics2011	Assist2017
Max length	50	200	128	128
Input size	101	201	1975	8235
Output size	50	200	987	4117
Hidden size	64	64	50	100
Skill layer size	5	20	61	102

Table 7.2: Hyperparameters used in DKT-IRT and SAKT-IRT on each dataset.

Method	Synth5	Sim200	Statics2011	Assist2017
DKT	0.803	0.838	0.793	0.731
SAKT	0.801	0.834	0.791	0.754
DKVMN	0.827	0.829	0.805	0.796
DKT-IRT	0.799	0.824	0.777	0.724
SAKT-IRT	0.798	0.833	0.775	0.728

Table 7.3: Test AUC values for various models on each dataset.

those reported in their respective literature [68, 73, 101].

7.2 Quantitative Results

As seen in Table 7.3, the two IRT-inspired knowledge tracing methods methods (DKT-IRT and SAKT-IRT) are able to produce AUC values competitive with other deep learning methods. As expected, the sacrifice in accuracy is smaller in simulated datasets. In Synth5 and Sim200, the responses were generated with known IRT models which match the architecture of IRT-inspired methods.

Note that when working on Synth5, we know that there were no discrimination parameters used to generate the data. As such, we fix all nonzero weights in the output layer to be equal to one by replacing Equation 6.4 with $W_p \leftarrow Q$. We do

not incorporate any estimation or knowledge of the guessing parameter c into the knowledge tracing model. This may account for a larger discrepancy in AUC between our methods and DKVMN in the Synth5 data than seen in Sim200.

When looking at the two real-world datasets, the trade-off in AUC is more significant, as it is not known if the observed student responses actually follow the ML2P model. There could also be inaccuracies in the given item-skill association Q-matrix, which our models are dependent on. Additional difficulties in the Assist2017 data (discussed in Section 7.1) concerning exercise-skill tags may explain the considerable performance gap between IRT-inspired methods and DKVMN on this dataset.

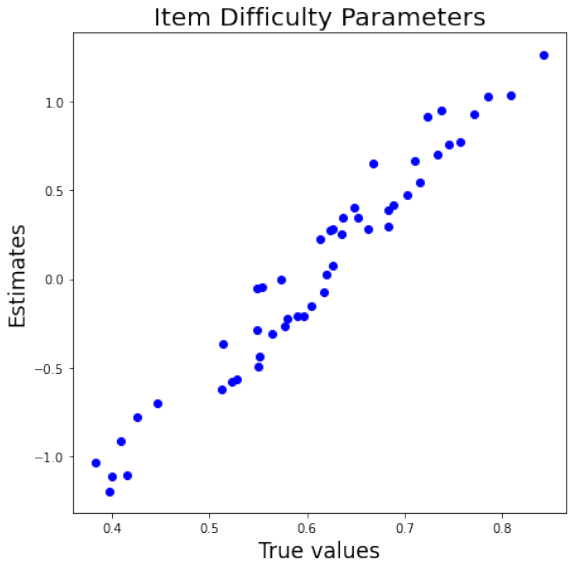

Figure 7.1: Correlation between DKT-IRT estimates and true values of Synth5 item difficulty. SAKT-IRT produced similar results.

A comparison between the output layer bias parameters and true item difficulty parameters is shown in Figure 7.1. This displays high correlation, and the trainable bias parameters in the output layer can be interpreted as approximations of the item difficulty parameters. The publicly available Synth5 data does not include access to the true values of student abilities $\mathbf{\Theta}$.

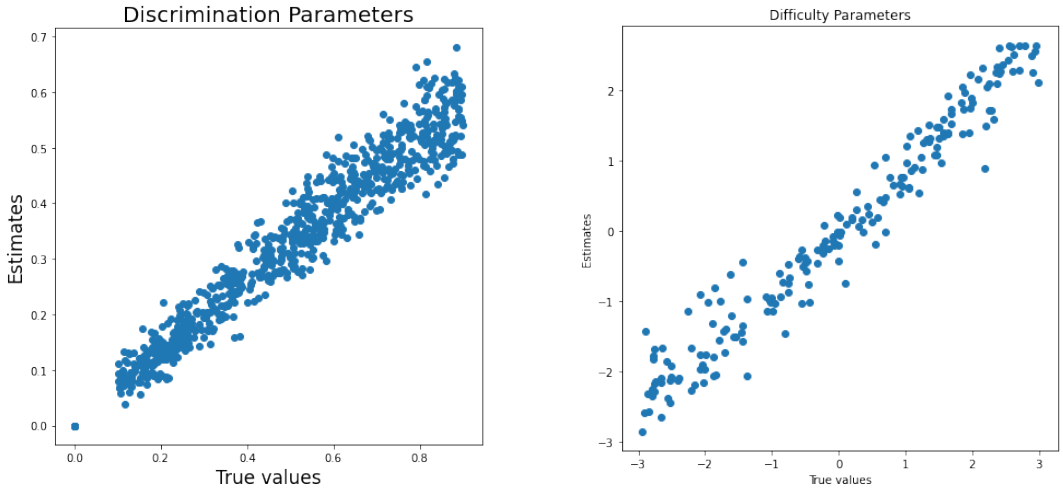

Figure 7.2: Correlation between true and estimated Sim200 item discrimination parameters (left), and item difficulty parameters (right).

The parameter estimates can be directly compared to the true parameters in the Sim200 dataset. In Figure 7.2, we can see the true values of item discrimination parameters a_{ik} and item difficulty parameter. The correlation here is very high, and the estimates for item parameters are quite accurate. The student ability parameters θ_{jk} plotted against estimates given by SAKT-IRT at the final timestep in Figure 7.3.

While there is a lot more noise in the student ability estimates, there is still significant correlation with the true values.

It is important to note that the estimates to $\boldsymbol{\Theta}$ do not require any additional computation or transformation and are directly obtained from a hidden neural network layer. This is an advantage over other deep knowledge tracing methods, which only output the probability of answering items correctly and require other methods of quantifying knowledge concepts. Recall that while estimates to student ability are the neuron activation values at the skill layer from feeding forward a response sequence, the discrimination parameter estimates are the trained weights connecting the skill layer to the output layer, visualized in Figure 6.1.

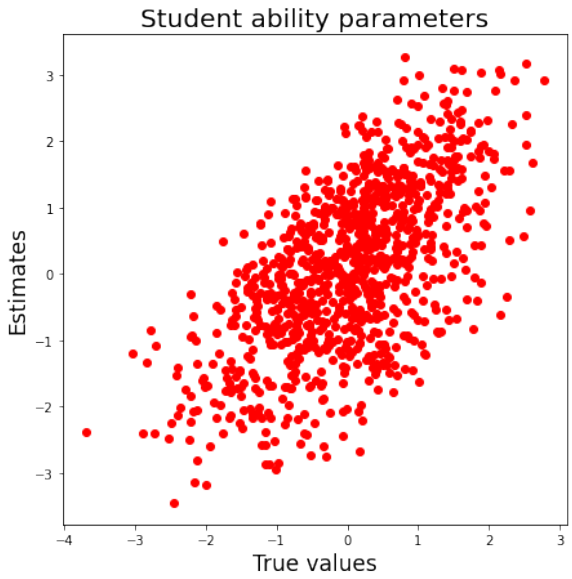

Figure 7.3: Correlation between true and estimated student ability parameters at $t = L = 200$ for the Sim200 dataset.

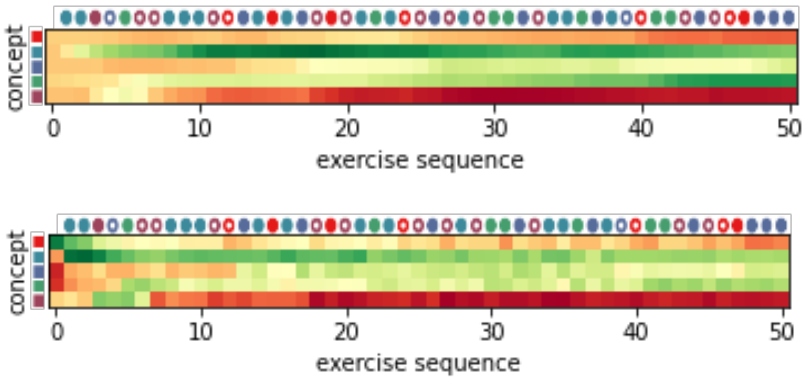

Figure 7.4: Tracing a student's knowledge mastery with DKT-IRT (top) and SAKT-IRT (bottom) as they progress through the items of the Synthetic-5 dataset.

The explicit representation of knowledge $\boldsymbol{\Theta}$ makes tracing student progress over time very convenient. For student j's response sequence of length L, IRT-inspired knowledge tracing methods return a $K \times (L+1)$ matrix, where the entry (k, t) gives the latent trait estimate to the k-th skill at time t, θ_{jkt}. This is visualized in Figure 7.4 on the Synth5 dataset. Notice how a correct response in a skill (filled-in circle) corresponds with a more green and less red skill value. Likewise, an incorrect response in a skill (hollow circle) corresponds with a more red skill or less green skill value. When comparing the knowledge tracing from DKT-IRT and SAKT-IRT, the DKT-IRT tracing is much more smooth as a student moves through the exam. This is likely because of the recurrent structure of an LSTM: the skill values at time t are only directly related to the values at time $t-1$. The SAKT-IRT tracing graphic is much choppier, because the attention mechanism does not have smooth recurrent structure and instead maintains connections to all previous interactions.

7.2.1 The Effect of Shuffled Responses

In Section 7.1, it was mentioned that the orderings of each student's responses on the simulated datasets were randomly permuted. If all students answer all questions in the same order, then the IRT-inspired knowledge tracing method suffers greatly, particularly when viewing the item parameter estimates. This is due to how student knowledge $\boldsymbol{\Theta}$ is estimated initially at $t = 0$, when no other information is provided to the model.

Consider the Sim200 dataset and assume that all students answer all items in the same order $(1, 2, \ldots, 199, 200)$. At $t = 0$ because of no prior information, all students have the same estimate for their latent traits, $\hat{\theta}_{jk0} = s^*_{0k}$ for all j as in Figure 6.1 – this is a poor estimate of student ability. This skill estimate is multiplied by the weight corresponding to a discrimination parameter of the first item, $w_{1k}q_{1k} = \hat{a}_{1k}$.

Since every student answers item 1 first, the same value s^*_{0k} is used in calculating $P(u_{1j} = 1)$ as in Equation 2.18 for all j. The inaccuracy of s^*_{0k} in all cases makes it difficult to estimate a_{1k} – the same is true for many items appearing early in the fixed response order. In Figure 7.5, we can compare the discrimination parameter estimates of items early versus late in the response order. Notice that the estimates are much more correlated with the true values for items towards the end of the sequence, since they have access to a more accurate estimate of each student's latent ability $\hat{\theta}_{jkt}$.

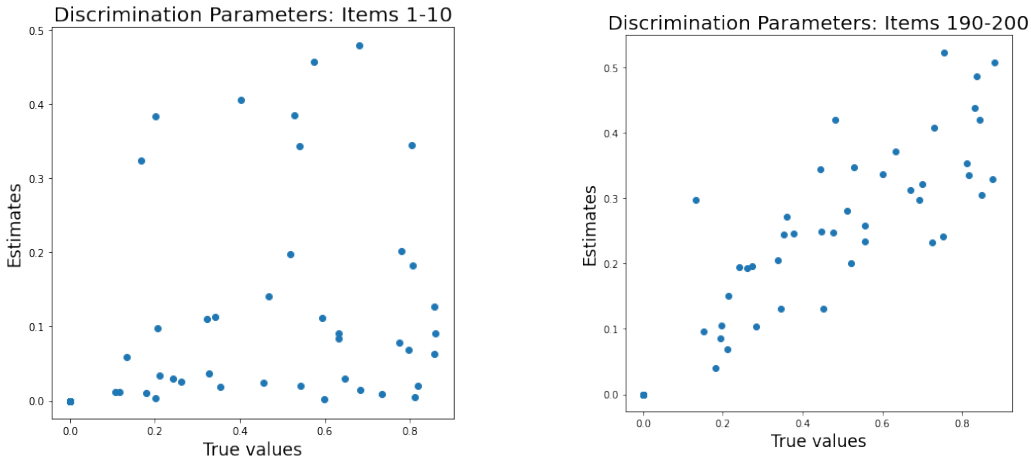

Figure 7.5: Discrimination parameter estimates of items early in the response sequence (left) and items late in the response sequence (right) when all students answer items in the same order.

7.2.2 Using Attention to Learn Item-Skill Associations

An advantage of other knowledge tracing methods such as DKT is that it does not require any expert annotation of the item-skill association. In other words, the only required data is the student response sequences, and not a Q-matrix. In the IRT-inspired knowledge tracing methods described in Chapter 6, this Q-matrix is required to build the architecture used for the results reported in Section 7.2.

Multiple methods of discovering item-skill relationships from deep models have been proposed in the literature. For example, Piech et al. [73] proposes calculating the conditional influence between each item for this task. This involves finding the relative change that a correct response on item i has on the probability of answering item j correctly, computed by feeding the corresponding response sequences through

DKT. In DKVMN [101], the correlation weights $\boldsymbol{w}_t$ from Equation 5.13 are used to quantify the relationship between items and memory slots. A similar approach is used by Pandey et al. in SAKT [68], using the attention weight in Equation 5.17 to relate each interaction in a response sequence.

We follow the framework of SAKT and use the attention mechanism to discover a Q-matrix. When using the SAKT-IRT architecture but removing the Q-matrix constraint, we arrive at a knowledge tracing framework that is nearly identical to SAKT. After training on the Synth5 dataset, we feed-forward a response sequence of a student answering all 50 items correctly and calculated the correlation weights similar to Equation 5.17:

$$\boldsymbol{c}_t = \text{softmax}\left(\frac{K\boldsymbol{q}_t}{\sqrt{d}}\right), \quad 0 \leq t \leq 50 \tag{7.2}$$

We can arrange all correlation vectors $\boldsymbol{c}_t$ into a single matrix $C \in \mathbb{R}^{51\times 51}$. A heat map of C is shown in Figure 7.6. Note that the upper-right half of C is all zero – this is due to masking out future interactions (interaction t cannot draw inferences from interaction $t+1$). The sum of each row of C is equal to 1, and each entry in row i can be interpreted as the relevance between each previous interaction and item i. For example, item 42 draws some information from previous interactions 2, 5, 10, 19, and 24, seen in row 42. In a similar manner, item 4 is heavily relied upon for reference when querying interactions 21, 29, 30, 35, 40, and 41, seen in column 4.

The natural assumption is that items which are correlated by Equation 7.2 measure the same latent concept. To better visualize the item-skill association, we construct a weighted graph G, with each node representing an item and edge weights

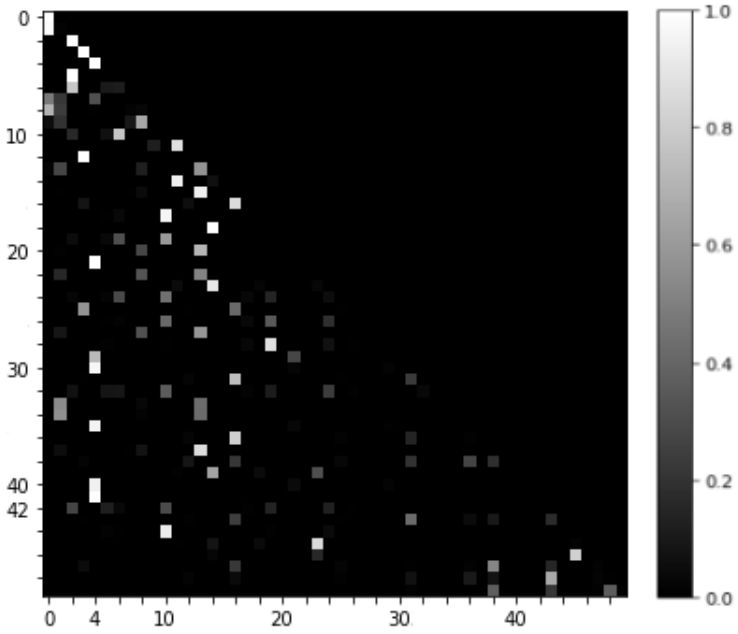

Figure 7.6: Heat map of the item correlation matrix C, where each row quantifies the relationship between an item and all previous items, starting in the top-left corner.

determined by the symmetric matrix $C + C^\top - \text{diag}(C)$. Using the NetworkX library for Python [41], we use a force-directed algorithm (`spring_layout`) to visualize G in 2-D space. This algorithm places node pairs with heavier weighted edges between them closer together [35], resulting in a clustering of items which are similar.

The graph G is shown in Figure 7.7. The different colors identify the true skill tag of each item – this information was provided to neither the neural network nor the graph visualization algorithm. NetworkX was not even provided the number of skills, yet still arranges the nodes into five distinct clusters. Notice that all items of similar skill are clustered together, displaying that the correlation matrix C in Figure 7.6 does in fact learn the item-skill associations. For example, items 42 and 4 are both

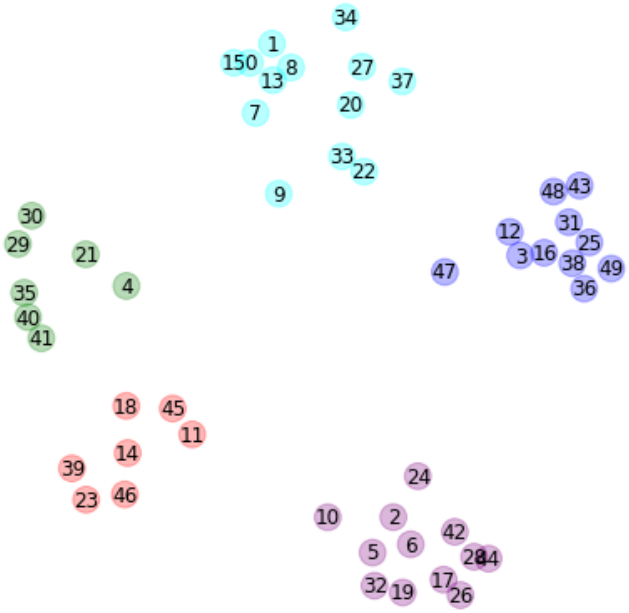

Figure 7.7: Visualization of the graph G, showing a five clusters of items which correspond to the five concepts in the Synth5 dataset.

found in clusters along with the items mentioned earlier which share larger values in C. The visualization of the graph G can be used to build a Q-matrix for use in DKT-IRT or SAKT-IRT, as described in Section 6.1.

7.3 Discussion

7.3.1 Future Extensions

The inclusion of Item Response Theory into the knowledge tracing framework gives significant advantages in terms of explainability and usability. But further work to increase AUC to the level of DKVMN while maintaining explainability is worth exploring, so that the trade-off leans more in favor of utilizing IRT.

Other ways to incorporate domain knowledge into specific knowledge tracing

models deserve further research as well. For example, SAKT uses the attention mechanism to calculated a correlation score with the current item and each previous item. But the information from the expert-annotated Q-matrix could be of use here: the correlation between items should only be significant if the two items share a common latent trait. So the item-skill association could be injected into Equation 7.2 in some way, either to mask out the relation between unrelated items (a constraint), or to encourage the matrix C in Figure 7.6 to more closely follow the known Q-matrix (a regularizer). A similar use of the Q-matrix could be incorporated into the framework of DKVMN in Equation 5.13.

In a more general extension, the idea of learning an d-dimensional representation of item interactions in knowledge tracing can be applied to static exams (i.e. Section 2.3). The input data to IRT parameter estimation methods, including ML2P-VAE from Chapter 3, is a single binary value for each item. Particularly for neural networks, this presents a difficultly for missing data – what if a student does not answer an item at all?

Though a common practice here is to classify a missing response as an incorrect response [93], the framework of knowledge tracing would allow for completely ignoring missing responses since it is designed to account for students who answer different sets of items. A practice similar to word2vec [61] could be useful in order to learn contextual information about a particular item response, rather than a simple binary indicator. Some adaptions from the dynamic environment to the fixed exam would be necessary, mainly removing the importance of the ordering of responses and time-

series. A model similar to SAKT-IRT, with positional encoding removed, would most readily fit the static assessment application.

7.3.2 Concluding Remarks

The connection between IRT and knowledge tracing presented in this thesis introduces a trade-off between accuracy and interpretability. Though IRT-inspired knowledge tracing does require an expert to annotate the item-skill association Q-matrix while other methods do not, explicitly incorporating this domain knowledge greatly increases the ability to interpret a deep learning model. Further, most intelligent tutoring systems provide an item-skill tag, so availability of the Q-matrix is not an unreasonable assumption.

The proposed method's ability to function as both a knowledge tracing model while also estimating item parameters gives it a unique interpretation rooted in Item Response Theory. This link with IRT is helpful in practice, because it provides an explicit and easy to obtain quantity for a student's latent abilities. The approximation of a student ability can be interpreted in the frame of IRT, as opposed to only a prediction of correctness for each item. This is a clear advantage that IRT-inspired knowledge tracing has over conventional non-interpretable deep learning methods in knowledge tracing.

CHAPTER 8
DISCUSSION

8.1 Applications Outside of Education

The ML2P-VAE method, introduced in Chapter 3 can be applied to areas outside of the realm of education. In general, it can be useful when some unobservable latent code is assumed to generate the observable data, and domain knowledge is available about said latent code. The ML2P-VAE method can be of use in most areas where other confirmatory IRT models are commonly implemented.

8.1.1 Health Sciences Questionnaires

In addition to education, Item Response Theory analysis has been performed on other psychometric questionnaires. For example, the Beck Depression Inventory (BDI) is a self-reported assessment that contains 21 items which aim to measure depressive traits in a clinical setting [5]. Questions ask respondents to choose one of four statements relating to one topic which best describes how they feel. For example, item 5 relates to "guilty feelings" and includes the statements

(0) I don't feel particularly guilty.

(1) I feel guilty over many things I have done or should have done.

(2) I feel quite guilty most of the time.

(3) I feel guilty all of the time.

After answering all 21 items, the BDI is scored by adding up the value of all selected responses. A total score in the range of 0–13 indicates minimal depressive

symptoms, the range 14–19 mild depression, scores between 20–28 indicate moderate depression, and any value above 29 is interpreted as severe depression [90]. Note the relation to classical test theory in the manner of scoring – IRT can give a more nuanced quantification of the latent traits associated with depression. Studies have theorized that the BDI is multidimensional, where two or three latent factors are influential in an examinee's observed responses. While different researchers have grouped BDI items by latent trait in various ways, some examples of the distinct traits include a "cognitive" factor, a "somatic" factor, and an "affective" factor [48].

Using the factor structure found by previous literature, a Q-matrix can be built relating the BDI items to depressive traits [34]. This domain-specific information can be utilized in ML2P-VAE to estimate the item and examinee parameters of IRT models. Note that the biggest advantage of ML2P-VAE, the capability of estimating a large number of latent traits, is not at play here. But the neural network structure of a VAE again does something that traditional IRT methods can not: incorporate examinee features other than item responses.

Previous studies have shown that demographic factors such as age and gender have a role in reported BDI responses and scores [27]. The ML2P-VAE framework can be slightly altered so that the input/output layer includes nodes not only for item responses, but for examinee features. If depressive symptoms are a function of BDI responses and demographic information, then it makes sense to build a VAE whose encoder maps responses and features to latent traits and whose decoder reconstructs the BDI data from said latent traits. Expanding the ML2P-VAE method to fit this

application is worthy of future research.

IRT has also been applied to personality questionnaires such as the Big Five and Myers-Briggs type indicator [62, 80]. These assessments aim to quantify traits such as extraversion, agreeableness, and judgement. Items are often presented as a statement with a Likert scale [55], where respondents select values 1 to 5 depending on how much they agree with the given statement. This format lines up well with Samejima's graded response model [85].

If the ML2P-VAE method is adapted for the graded response model as described in Section 4.3.1, then the neural network approach to latent trait analysis could be useful. Even without an extension to Samejima's model, ML2P-VAE could be applied to personality questionnaires by formatting the Likert scale data into dichotomous responses – ignoring the "*strongly* agree/disagree" distinction, and imposing cut-off between "agree" and "disagree" responses.

8.1.2 Sports Analytics

In this section, we summarize the publication "Variational Autoencoders for Baseball Player Evaluation," [20] presented at the International Conference on Fuzzy Systems and Data Mining, 2019. In this work, the idea of latent traits influencing observable performance is extended from educational measurement to professional sports, specifically player evaluation in baseball. Though there does not exist any formal theory relating to latent athletic skills such as what education has in IRT, parallels can still be drawn between the two fields.

This work focuses on professional baseball players. In education, the "observable performance" was given by binary responses to questions on exams – here, the observable data is recorded statistics from a player during a particular season, such as hits, stolen bases, home runs, or strikeouts. Where IRT focused on estimating student's latent knowledge $\boldsymbol{\Theta}$, the goal of this work is to develop *new* measures to quantify a baseball player's skills on the field, which influence what is observed in the box score.

The ideas of the ML2P-VAE model can be used for this task. We first define four underlying skills required for baseball players to be successful: *contact* (how often does the batter hit the ball), *power* (how hard does the player hit the ball), *base running* (is the player good at running the bases), and *pitch intuition* (does the player swing when they are supposed to). In ML2P-VAE, the item-skill relationship given by the Q-matrix is instrumental in allowing interpretation of a hidden neural network layer. A similar binary matrix is developed in the baseball application, relating the four skills to the observed data. For example, the statistic "home runs" requires only the power skill, while avoiding "strikeouts" requires both contact and pitch intuition.

After training the modified VAE, we can obtain quantities for every baseball player in each of the four skills. In order to evaluate the effectiveness of our skill values, we compare the skills outputted by the VAE encoder to the commonly used baseball statistics such as contact rate, speed score, isolated power, and on-base percentage [56]. We don't strive to exactly replicate these statistics (since we are developing *new* measures), but it can be helpful to reference in order to confirm that our measures

actually represent what we intend them to quantify. For example, the players with the largest *power* value include Barry Bonds and Mark McGwire (notable power hitters), and the players with the highest *base running* value are Rickey Henderson and Lou Brock (famous base-stealers).

Though there is less mathematical theory supporting the application of ML2P-VAE to the sports analytics field, it still yields interesting results. We are able to interpret a hidden layer of the neural network, and use those values to quantify the latent code's influence on observable performance. The proposed measures of underlying athletic skills can be useful for sports franchises in evaluating a player's usefulness to a team.

8.2 Conclusion

In this thesis, novel neural network methods for application in educational measurement are presented and described in detail. Even outside of the application area, the work is interesting from a machine learning point of view, as modifications to neural architecture are made which allow for direct interpretation of trainable parameters and a hidden layer.

The primary research described in Chapter 3 details the ML2P-VAE method for IRT parameter estimation. Using domain-specific knowledge of the Q-matrix detailing the binary relationship between items and skills, the weights matrix of the VAE decoder is constrained. This causes the generative model $p_\beta(\boldsymbol{x}|\boldsymbol{z})$ from Section 2.2.2 to match the exact form of the ML2P model in Equation 2.22. After train-

ing ML2P-VAE, the VAE decoder becomes an approximate ML2P model, and the encoder maps a student's binary responses to an estimate of the latent ability $\boldsymbol{\Theta}$. At the very least, the ML2P-VAE method is an interesting and unorthodox use of a variational autoencoder which yields comparable accuracy to that of more traditional IRT methods. But additionally, it holds a considerable practical advantage over these other methods in the context of big data and computational complexity.

The ML2P-VAE technique avoids the curse of dimensionality, a significant problem faced by traditional IRT parameter estimation methods explored in Section 2.4.2.1. Since most other methods rely on numerical integration or MCMC methods, estimating parameters where the dimension of the latent traits $\boldsymbol{\Theta} \in \mathbb{R}^K$ is larger than $K = 10$ is difficult, and quickly becomes unobtainable as K increases. Rather than computing a K-dimensional integral, ML2P-VAE uses a neural network to learn a K-dimensional posterior distribution. In the field of deep learning, the scale of $K \in [3, 20]$ (latent trait dimensionality explored in this thesis) does not present any computational difficulty.

In addition to scalability within the area of educational measurement, the ML2P-VAE method provides a novel VAE architecture which allows for correlated latent code. Though most VAE implementations assume that $\boldsymbol{z} \sim \mathcal{N}(0, I)$, if domain knowledge about the latent space $\boldsymbol{z} \in \mathbb{R}^K$ is available, then the proposed neural architecture can fit a VAE to the more general multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$. As discussed in Section 3.1.1, this is non-trivial and we prove that for any input $\boldsymbol{x}_0 \in \mathbb{R}^n$, the proposed VAE encoder will output a posterior probability

distribution $q_\alpha(\boldsymbol{z}|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{\mu}_0, \Sigma_0)$. The VAE design presented in this thesis is of interest in other fields which wish to impose additional structure on the latent code.

The second part of this thesis extends the ideas from ML2P-VAE to the knowledge tracing problem, where the learning environment is much more dynamic. While deep knowledge tracing models are capable of predicting student success on their next interaction, this computation is not explainable. The work in this thesis suggests incorporating Item Response Theory into the knowledge tracing framework in order to provide an explicit representation of student knowledge $\boldsymbol{\Theta}$ over time.

The IRT-inspired knowledge tracing methods can be integrated directly into many other knowledge tracing models, including DKT [73] and SAKT [68]. Similar to ML2P-VAE, the core idea is to inject domain knowledge into the neural architecture to obtain an approximate IRT model. This presents a trade-off between prediction power and interpretability, while remaining competitive with state-of-the-art models in terms of test AUC scores.

In both parts of this thesis research, there is room for future work. ML2P-VAE can be extended to other IRT models to include a guessing parameter or a graded response model, discussed in Section 4.3.1. The notion of incorporating expert annotation into VAE architecture can be applied to areas in the health sciences, described in Section 8.1. Section 7.3.1 describes additional paths to inject IRT into the framework of deep knowledge tracing. The use of deep learning methods in education is becoming more popular, and the work in this thesis explores a few of many avenues for potential research. The contributions detailed here stand on their own in both

the studies of machine learning and educational measurement, and identify interesting parallels between the two fields.

APPENDIX A
ARTIFICIAL NEURAL NETWORKS

Artificial Neural Networks (ANN) are commonly understood to be complicated black box machine learning methods which produce high levels of accuracy, while sacrificing interpretability [9]. This assessment is true in part – the end-to-end decision process of a trained neural network is very convoluted. But after zooming in to the inner workings of an ANN, each individual part is quite simple.

A.1 Architecture

Neural networks have a graph-like structure with vertices (nodes) and weighted edges. A basic feed-forward neural network (FFN) consists of an input layer, a number of hidden layers, and an output layer. A "layer" l consists of n_l nodes, and each node i is densely connected to every node in the previous layer $l-1$ by a weighted edge. In this manner, the subgraph containing all nodes of layer $l-1$ and all nodes of layer l can be seen as a complete bipartite graph K_{n_{l-1},n_l}. A simple FFN is shown in Figure A.1 which takes inputs with 10 features and classifies into three categories [54]. For example, inputs could represent the weight, height, hair length, etc. of a pet, with the task of classifying the pet as a cat, dog, or bird [64].

While the architecture of a neural network can be described using the lens of graph theory, the inner workings are better explained using basic linear algebra. Each layer acts as a function from $\mathbb{R}^{n_{l-1}}$ to $\mathbb{R}^{n_l}$. Specifically, this function is a linear transformation followed by a nonlinear rescaling. For a given input vector $\boldsymbol{x}^* \in \mathbb{R}^{n_0}$

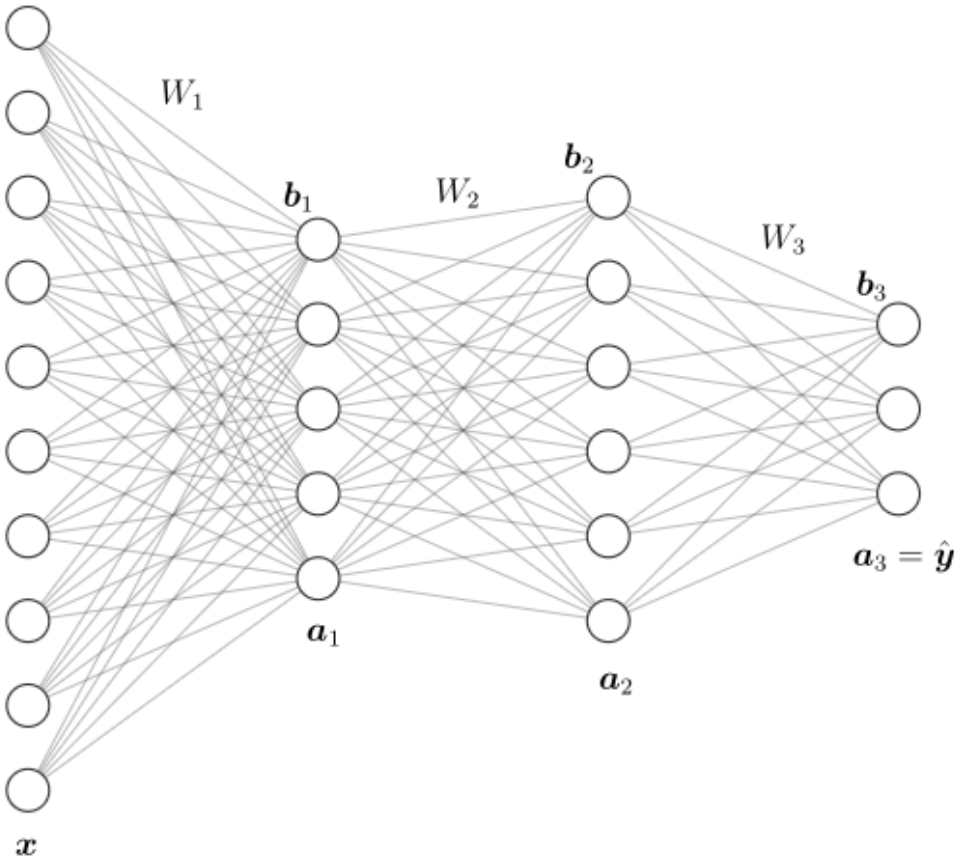

Figure A.1: A basic FFN with input size $n_0 = 10$ and output size $n_3 = 3$, with two hidden layers of size $n_1 = 5$ and $n_2 = 6$.

and its corresponding true label $\boldsymbol{y}^*$, the value in the first hidden layer is calculated as

$$\boldsymbol{a}_1^* = f_1(W_1\boldsymbol{x}^* + \boldsymbol{b}_1) \tag{A.1}$$

where $W_1 \in \mathbb{R}^{n_1 \times n_0}$ and $\boldsymbol{b}_1 \in \mathbb{R}^{n_1}$ are the trainable *weights* matrix and *bias* vector. The notion of "trainable" parameters is explored further in Appendix A.3. The value $\boldsymbol{a}_1^*$ is called the *activation* value of the input $\boldsymbol{x}^*$ at hidden layer $l = 1$. The non-decreasing function f_1 is called an *activation function* which applies a (possibly) non-linear rescaling to the vector $\boldsymbol{z}_1^* = W_1\boldsymbol{x}^* + \boldsymbol{b}_1 \in \mathbb{R}^{n_1}$ element-wise. Examples of

different activation functions are given in Appendix A.2. Matrix notation can also be abandoned by writing the activation of the i-th node in layer $l = 1$ as

$$a^*_{1,i} = f_1\left(b_{1,i} + \sum_{j=1}^{n_0} w^1_{ij} \cdot x^*_j\right) \tag{A.2}$$

where w^1_{ij} is the element in the i-th row and j-th column of W_1, the weight connecting the j-th node of layer 0 to the i-th node of layer 1.

The activation value of the input $\boldsymbol{x}^*$ at hidden layer $l = 2$ and the output layer $l = 3$ can similarly be computed as

$$\begin{aligned} \boldsymbol{a}^*_2 &= f_2(W_2\boldsymbol{a}^*_1 + \boldsymbol{b}_2) \\ \hat{\boldsymbol{y}}^* &= \boldsymbol{a}^*_3 = f_3(W_3\boldsymbol{a}^*_2 + \boldsymbol{b}_3) \end{aligned} \tag{A.3}$$

where the weights matrices $W_2 \in \mathbb{R}^{n_2 \times n_1}$ and $W_3 \in \mathbb{R}^{n_3 \times n_2}$ and bias vectors $\boldsymbol{b}_2 \in \mathbb{R}^{n_2}$ and $\boldsymbol{b}_3 \in \mathbb{R}^{n_3}$ are trainable. Before training, all trainable parameters are typically initialized randomly. The output value $\hat{\boldsymbol{y}}^*$ serves as the prediction for input $\boldsymbol{x}^*$.

For classification, the true label $\boldsymbol{y}$ is often a one-hot encoding, so the prediction $\hat{\boldsymbol{y}}^*$ should be a probability distribution each entry describes the certainty of the model in classifying the input $\boldsymbol{x}^*$ as each possible class. Returning to the earlier example, an input with features pertaining to a cat has the true label $(1, 0, 0)$. An output prediction may give $(0.65, 0.32, 0.03)$, meaning that the neural network is 65% confident that the input features are that of a cat, 32% confident that the input is a dog, and 3% confident that the input is a bird.

A.2 Activation Functions

The main purpose of activation functions is to rescale each layer so that every activation value falls in the same range. Performing many matrix multiplications in a row can easily cause values to become very large, leading to over-fitting and other complications [88]. It can be helpful to use an activation function to map values to the range of $(0, 1)$ because of the effect of the numbers 0 and 1 in multiplication, map to $(-1, 1)$ to utilize positive and negative values, or map to $[0, \infty)$ to avoid negative values.

Though custom activation functions can be defined and easily implemented, below are a few examples of popular activation functions used in neural networks [1] [2]. Each of these are independently applied to a vector element-wise, except for the softmax function which maps an n-dimensional vector to a discrete probability distribution with n possible values. Denote $z = \sum_k a_k w_k + b$ as the input to a node's activation function.

$$\text{Sigmoid: } \sigma(z) = \frac{1}{1 + e^{-z}} \qquad \mathbb{R} \to (0, 1) \tag{A.4}$$

The sigmoid activation function has the form of the logistic curve and maps values to be between 0 and 1.

$$\text{Hyperbolic tangent: } \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \qquad \mathbb{R} \to (-1, 1) \tag{A.5}$$

Hyperbolic tangent has a similar curvature to that of the sigmoid, but maps values to between -1 and 1.

$$\text{Rectified Linear Unit (ReLU): } \text{relu}(z) = \max\{0, z\} \qquad \mathbb{R} \to (0, \infty) \tag{A.6}$$

The ReLU function is often used to combat the "learning slowdown" problem that the sigmoid and tanh can face – if an input z_0 is very large or very small, then the derivative $\left.\frac{d\sigma}{dz}\right|_{z=z_0}$ is very small, causing gradient descent iterations to improve slowly. The derivative of the ReLU function is either exactly 0 or exactly 1, which helps speed up the training process.

$$\text{Softmax: } \text{softmax}(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \qquad \mathbb{R}^n \to \{P(x = i)\}_{i=1}^{n} \tag{A.7}$$

When using the softmax activation function, the activation a single node is a function of the activation of all other nodes within the same layer. This is seen in the summation over n nodes in Equation A.7. It is also straightforward to see that for any input, the sum of all n nodes in the layer always adds up to exactly 1 – so a softmax layer with n nodes can be understood as a discrete probability distribution with n classes. This can be particularly useful in the output layer of a multi-class classification neural network model.

A.3 Optimization and Backpropagation

Figure A.1 and Equations A.1 and A.3 show that given an input $\boldsymbol{x}^*$, a feed-forward neural network can output a prediction $\hat{\boldsymbol{y}}^*$. We now turn to the way in which $\hat{\boldsymbol{y}}^*$ serves as a *quality* prediction of the true value $\boldsymbol{y}^*$. The terminology of "training" a neural network refers to finding optimal settings of the weights matrices W_l and bias

vectors $\boldsymbol{b}_l$ in each layer l which minimize the error between predictions $\hat{\boldsymbol{y}}^*$ and true inputs $\boldsymbol{y}^*$. Such measures of error are called *loss functions*.

Though there are many candidates for loss functions such as cross-entropy (see Equation 2.2) or hinge loss [36], consider the simple mean squared-error loss function

$$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = ||\boldsymbol{y} - \hat{\boldsymbol{y}}||_2^2 = \frac{1}{K}\sum_{k=1}^{K}(y_k - \hat{y}_k)^2 \tag{A.8}$$

where K is the dimension of the target (i.e. the number of output nodes).

Recall that if an input (or set of inputs) $\boldsymbol{x}^*$ is fixed, then the prediction $\hat{\boldsymbol{y}}^*$ outputted by the neural network is a function of the weights and biases W_l and $\boldsymbol{b}_l$. As such, we can compute partial derivatives of $\mathcal{L}$ with respect to each trainable parameter and use a gradient descent algorithm to minimize Equation A.8.

Though deep neural networks can have thousands, millions, or billions of parameters [12], calculating gradients remains feasible because of the backpropagation algorithm [82]. While obtaining predictions from a neural networks works in a left-to-right fashion ($\boldsymbol{a}_3$ depends on $\boldsymbol{a}_2$, which depends on $\boldsymbol{a}_1$, which depends on $\boldsymbol{x}$), gradient calculations are computed right-to-left. This is due to the role of the chain rule.

First consider calculating the partial derivative of a particular weight in the final layer, w_{ij}^3. Denote the input to an activation function at layer l as $\boldsymbol{z}_l = W_l\boldsymbol{a}_{l-1}+b_l$ so that $\boldsymbol{a}_l = f_l(\boldsymbol{z}_l)$. Using Equation A.3, we can write

$$\frac{\partial\mathcal{L}(\boldsymbol{y}, \boldsymbol{a}_3)}{\partial w_{ij}^3} = \frac{\partial\mathcal{L}(\boldsymbol{y}, \boldsymbol{a}_3)}{\partial a_{3,i}} \cdot \frac{\partial a_{3,i}}{\partial z_{3,i}} \cdot \frac{\partial z_{3,i}}{\partial w_{ij}^3} = \frac{\partial\mathcal{L}(\boldsymbol{y}, \boldsymbol{a}_3)}{\partial a_{3,i}} \cdot f_3'(z_{3,i}) \cdot a_{2,j} \tag{A.9}$$

Now consider the change in loss with respect to a trainable parameter in the second-to-last layer. Choose a weight w_{jk}^2 whose right endpoint is the same node as the left

endpoint of w_{ij}^3 used in Equation A.9. We have

$$\begin{aligned}\frac{\partial \mathcal{L}(\boldsymbol{y}, \boldsymbol{a}_3)}{\partial w_{kj}^2} &= \sum_{i=1}^{K} \left(\frac{\partial \mathcal{L}(\boldsymbol{y}, \boldsymbol{a}_3)}{\partial a_{3,i}} \cdot \frac{\partial a_{3,i}}{\partial z_{3,i}} \cdot \frac{\partial z_{3,i}}{\partial a_{2,j}} \right) \cdot \frac{\partial a_{2,j}}{\partial z_{2,j}} \cdot \frac{\partial z_{2,j}}{\partial w_{jk}^2} \\ &= \sum_{i=1}^{K} \left(\frac{\partial \mathcal{L}(\boldsymbol{y}, \boldsymbol{a}_3)}{\partial a_{3,i}} \cdot f_3'(z_{3,i}) \cdot w_{ij}^3 \right) \cdot f_2'(z_{2,j}) \cdot a_{1,k}\end{aligned} \tag{A.10}$$

Notice how in Equation A.10, information first calculated in Equation A.9 is re-used. Particularly, the partial derivative of a parameter found in layer l is a sum of partial derivatives of values found in layer $l+1$. In this sense, the backpropagation algorithm works right-to-left; first calculating values in layer L that will later be used in all layers $l < L$.

After the gradient of $\mathcal{L}$ is found using backpropagation, a gradient descent update is performed for an input $\boldsymbol{x}$:

$$\boldsymbol{\Lambda}_{t+1} \leftarrow \boldsymbol{\Lambda}_t - \eta \nabla_{\boldsymbol{\Lambda}} \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) \tag{A.11}$$

where $\boldsymbol{\Lambda}$ is a vector containing all trainable parameters W_l and $\boldsymbol{b}_l$ and η is the learning rate hyperparameter [84]. This process is repeated iteratively using randomly selected inputs $\boldsymbol{x}$ from the training set until convergence, as in the stochastic gradient descent (SGD) method.

APPENDIX B
ML2PVAE PACKAGE DETAILS

In this section, we provide a tutorial of the **ML2Pvae** software package for R. Functions and data which are exported by the package are listed in `blue`. This tutorial uses a simulated dataset which is accessible through the package, including:

- A Q-matrix (`q_matrix`) relating $n = 30$ items to $K = 3$ latent skills
- A covariance matrix (`correlation_matrix`) detailing the correlations between the latent skills
- A set of $N = 5,000$ binary responses (`responses`) to $n = 30$ items, generated by the ML2P model with true parameters:
  - $\boldsymbol{\Theta}_j \in \mathbb{R}^3$, $1 \leq j \leq 5,000$ (`theta_true`)
  - $\boldsymbol{a}_i \in \mathbb{R}^3$, $1 \leq i \leq 30$ (`disc_true`)
  - $b_i \in \mathbb{R}$, $1 \leq i \leq 30$ (`diff_true`)

The **ML2Pvae** package has five easy-to-use functions to assist in building, training, and evaluating ML2P-VAE models. The functions `build_vae_independent()` and `build_vae_correlated()` construct a modified VAE architecture as specified by the user. The former assumes that the latent traits are independent ($\boldsymbol{\Theta} \sim \mathcal{N}(0, I)$), and the latter assumes knowledge of correlated latent traits ($\boldsymbol{\Theta} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$), as described in Section 3.1.1.

To train a VAE on a dataset, the function `train_model()` can be used. After the model has been fitted, the function `get_item_parameter_estimates()` grabs the

relevant trainable weights from the decoder which serve as estimates of $\boldsymbol{a}_i$ and b_i. The function `get_ability_parameter_estimates()` feeds student responses through the encoder to obtain estimates to $\boldsymbol{\Theta}_j$.

The functionality of **ML2Pvae** is displayed below. Note that while the neural network models are created with Tensorflow and Keras [2] inside of **ML2Pvae**, those packages are not employed by the user. This is by design in order to make the ML2P-VAE method accessible to IRT researchers in psychometrics who may not have knowledge of neural networks. Further explanation and documentation for **ML2Pvae** can be found at https://cran.r-project.org/web/packages/ML2Pvae. Source code of the software is found at https://github.com/converseg/ML2Pvae.

B.1 Software Tutorial

```
# Install package from CRAN or GitHub
install.packages('ML2Pvae')
devtools::install_github('converseg/ML2Pvae')

# Load package
library(ML2Pvae)

# Load sample data (included in package)
data <- as.matrix(responses)
Q <- as.matrix(q_matrix)
cov_matrix <- as.matrix(correlation_matrix)

# Model parameters
num_items <- as.double(dim(Q)[2])
num_skills <- as.double(dim(Q)[1])
num_students <- dim(data)[1]
enc_arch <- c(16L, 8L)
enc_act <- c('relu', 'tanh')
out_act <- 'sigmoid'

```

```
# Build ML2P-VAE assuming known correlation between latent traits
models <- build_vae_correlated(
  num_items,
  num_skills,
  Q,
  covariance_matrix = cov_matrix,
  model_type = 2,
  enc_hid_arch = enc_arch,
  hid_enc_activation = enc_act,
  output_activation = out_act)
encoder <- models[[1]]
decoder <- models[[2]]
vae <- models[[3]]

# Training parameters
num_train <- floor(0.8 * num_students)
num_test <- num_students - num_train
data_train <- data[1:num_train,]
data_test <- data[(num_train + 1):num_students,]
num_epochs <- 15
batch_size <- 8

# Train neural network
history <- train_model(
  vae,
  data_train,
  num_epochs = num_epochs,
  batch_size = batch_size,
  verbose = 1)

# Get IRT parameter estimates
item_param_estimates <- get_item_parameter_estimates(
  decoder, model_type = 2)
diff_est <- item_param_estimates[[1]]
disc_est <- item_param_estimates[[2]]
test_theta_est <- get_ability_parameter_estimates(
  encoder, data_test)[[1]]

# Load in true values (included in package)
true_diff <- as.matrix(diff_true)
true_disc <- as.matrix(disc_true)
true_theta<- as.matrix(theta_true)
true_theta_test <- theta_params_true[(num_train+1):num_students,]

```

```r
# Plot estimates against true values
plot(true_diff, diff_est, pch = '*',
  main = 'Difficulty Parameters',
  xlab = 'True', ylab = 'Estimated')
matplot(t(true_disc), t(disc_est), pch = '*',
  main = 'Discrimination Parameters',
  xlab = 'True', ylab = 'Estimated')
matplot(true_theta_test, test_theta_est, pch = '*',
  main = 'Ability Parameters',
  xlab = 'True', ylab = 'Estimated')
```

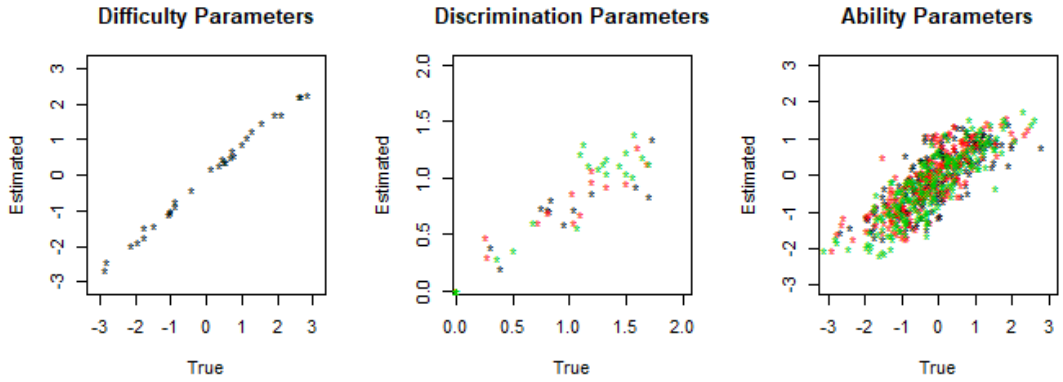

Figure B.1: Correlation plots of IRT parameter estimates produced by the above code tutorial.

REFERENCES

[1] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] JJ Allaire and François Chollet. *keras: R Interface to Keras*, 2020. R package version 2.3.0.0.9000.

[3] Kendall Atkinson. *An Introduction to Numerical Analysis*. John Wiley and Sons, 1989.

[4] F. Baker and S. Kim. *Item Response Theory Parameter Estimation Techniques*. Taylor & Francis Group, 2nd edition, 2004.

[5] Aaron T Beck, Robert A Steer, and Gregory Brown. Beck depression inventory–ii. *Psychological Assessment*, 1996.

[6] Adi Ben-Israel. A newton-raphson method for the solution of systems of equations. *Journal of Mathematical analysis and applications*, 15(2):243–252, 1966.

[7] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[8] Allen Birnbaum. Some latent trait models and their use in inferring an examinee's ability. In F. Lord and M. R. Novick, editors, *Statistical Theories of Mental Test Scores*, pages 397–479. Addison-Wesley, 1968.

[9] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[10] D.M. Blei, A. Kucukelbir, and J.D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

[11] R. D. Bock and M. Aitkin. Marginal maximum likelihood estimation of item parameters: Application of an EM algorithm. *Psychometrika*, 46(4):443–459, 1981.

[12] Tom Brown et al. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[13] Li Cai. High-dimensional exploratory item factor analysis by a Metropolis–Hastings Robbins–Monro algorithm. *Psychometrika*, 75(1):33–57, 2009.

[14] Li Cai. Metropolis-Hastings Robbins-Monro algorithm for confirmatory item factor analysis. *Journal of Educational and Behavioral Statistics*, 35(3):307–335, 2010.

[15] R. Philip Chalmers. mirt: A multidimensional item response theory package for the R environment. *Journal of Statistical Software*, 48(6):1–29, 2012.

[16] Yunxiao Chen, Xiaoou Li, and Siliang Zhang. Joint maximum likelihood estimation for high-dimensional exploratory item factor analysis. *Psychometrika*, 84:124–146, 2019.

[17] Aaron Chou. What's in the "black box"? Balancing financial inclusion and privacy in digital consumer lending. *Duke Law Journal*, 69:1183–1217, 2020.

[18] G. Converse, M. Curi, and S. Oliveira. Autoencoders for educational assessment. In Seiji Isotani, Eva Millán, Amy Ogan, Peter Hastings, Bruce McLaren, and Rose Luckin, editors, *Artificial Intelligence in Education*, pages 41–45. Springer International Publishing, 2019.

[19] Geoffrey Converse. *ML2Pvae: Variational Autoencoder Models for IRT Parameter Estimation*, 2020. R package version 1.0.0.

[20] Geoffrey Converse, Brooke Arnold, Mariana Curi, and Suely Oliveira. Variational autoencoders for baseball player evaluation. In *Fuzzy Systems and Data Mining V - Proceedings of FSDM 2019*, volume 320 of *Frontiers in Artificial Intelligence and Applications*, pages 305–311. IOS Press, 2019.

[21] Geoffrey Converse, Mariana Curi, Suely Oliveira, and Jonathan Templin. Estimation of multidimensional item response theory models with correlated latent variables using variational autoencoders. *Machine Learning*, 2021.

[22] Geoffrey Converse, Suely Oliveira, and Shi Pu. Incorporating item response theory into knowledge tracing. In Ido Roll, Danielle McNamara, Sergey Sosnovsky, Rose Luckin, and Vania Dimitrova, editors, *Artificial Intelligence in Education*, pages 114–118. Springer International Publishing, 2021.

[23] Albert Corbett and John Anderson. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, 4:253–278, 1995.

[24] M. Curi, G. Converse, J. Hajewski, and S. Oliveira. Interpretable variational autoencoders for cognitive models. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019.

[25] M.A. da Silva, R. Liu, A.C. Huggins-Manley, and J.L. Bazan. Incorporating the q-matrix into multidimensional item response theory models. *Educational and Psychological Measurement*, 2018.

[26] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, abs/1901.02860, 2019.

[27] Antonio Reis de Sá Junior, Graziela Liebel, Arthur Guerra de Andrade, Laura Helena Andrade, Clarice Gorenstein, and Yuan-Pang Wang. Can gender and age impact on response pattern of depressive symptoms among college students? a differential item functioning analysis. *Frontiers in psychiatry*, 10:50, 2019.

[28] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[29] J. Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

[30] Carl Doersch. Tutorial on variational autoencoders. *arXiv:1606.05908*, 2016.

[31] Samira Ebrahimi Kahou, Vincent Michalski, Kishore Konda, Roland Memisevic, and Christopher Pal. Recurrent neural networks for emotion recognition in video. In *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, page 467–474, 2015.

[32] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[33] Y. Feng, B. T. Habing, and A. Huebner. Parameter estimation of the reduced RUM using the EM algorithm. *Applied Psychological Measurement*, 38:137–150, 2014.

[34] T.M. Fragoso and M. Curi. Improving psychometric assessment of the Beck depression inventory using multidimensional item response theory. *Biometrical Journal*, pages 527–540, 2013.

[35] Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

[36] Claudio Gentile and Manfred KK Warmuth. Linear hinge loss and average margin. *Advances in neural information processing systems*, 11:225–231, 1998.

[37] William Gilpin. Deep learning of dynamical attractors from time series measurements. In *Conference on Neural Information Processing Systems*, 2020.

[38] Randy Goebel, Ajay Chander, Katharina Holzinger, Freddy Lecue, Zeynep Akata, Simone Stumpf, Peter Kieseberg, and Andreas Holzinger. Explainable AI: the new 42? In *International cross-domain conference for machine learning and knowledge extraction*, pages 295–303. Springer, 2018.

[39] Q. Guo, M. Cutumisu, and Y. Cui. A neural network approach to estimate student skill mastery in cognitive diagnostic assessments. In *10th International Conference on Educational Data Mining*, 2017.

[40] Shelby J. Haberman. Identifiability of parameters in item response models with unconstrained ability distributions. Technical Report RR-05-24, Research and Development, ETS, December 2005.

[41] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[42] D. Hammerstrom. Neural networks at work. *IEEE Spectrum*, 30(6):26–32, 1993.

[43] Kyung (Chris) T. Han and Insu Paek. A review of commercial software packages for multidimensional IRT modeling. *Applied Psychological Measurement*, 38(6):486–498, 2014.

[44] Jiangang Hao, Lei Liu, Alina A von Davier, Nathan Lederer, Diego Zapata-Rivera, Peter Jakl, and Michael Bakkenson. Epcal: Ets platform for collaborative assessment and learning. *ETS Research Report Series*, 2017(1):1–14, 2017.

[45] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[46] R. Henson and J. Templin. Large-scale language assessment using cognitive diagnosis models. In *Annual meeting of the National Council for Measurement in Education*, 2007.

[47] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[48] Chiungjung Huang and Jyun-Hong Chen. Meta-analysis of the factor structures of the Beck depression inventory–II. *Assessment*, 22(4):459–472, 2015.

[49] Edwin T Jaynes. Information theory and statistical mechanics. *Physical review*, 106(4):620, 1957.

[50] Zhehan Jiang and Jonathan Templin. Gibbs samplers for logistic item response models via the pólya–gamma distribution: A computationally efficient data-augmentation strategy. *Psychometrika*, 84:358–374, 2018.

[51] D. Kingma and M. Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations*, 2014.

[52] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[53] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, page 79–86, 1951.

[54] Alexander LeNail. Nn-svg: Publication-ready neural network architecture schematics. *Journal of Open Source Software*, 4(33):747, 2019.

[55] Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.

[56] Sports Reference LLC. Baseball-reference batting statistics. Baseball-Reference.com, 2021.

[57] F. Lord and M. R Novick. *Statistical theories of mental test scores*. IAP, 1968.

[58] Geoff N Masters. A Rasch model for partial credit scoring. *Psychometrika*, 47(2):149–174, 1982.

[59] Rob R Meijer and Michael L Nering. Computerized adaptive testing: Overview and introduction. *Applied psychological measurement*, 23(3):187–194, 1999.

[60] Xiao-Li Meng and Stephen Schilling. Fitting full-information item factor models and an empirical investigation of bridge sampling. *Journal of the American Statistical Association*, 91(435):1254–1267, 1996.

[61] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, page 3111–3119, Red Hook, NY, USA, 2013. Curran Associates Inc.

[62] Isabel Briggs Myers. The Myers-Briggs type indicator: Manual (1962). 1962.

[63] In Jae Myung. Tutorial on maximum likelihood estimation. *Journal of Mathematical Psychology*, 47(1):90–100, 2003.

[64] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[65] The United States Department of Justice. The equal credit opportunity act, Jul 2020.

[66] Christopher Olah. Understanding LSTM networks, 2015.

[67] James Ong and Sowmya Ramachandran. Intelligent tutoring systems: Using ai to improve training performance and roi. *Networker Newsletter*, 19(6):1–6, 2003.

[68] Shalini Pandey and George Karypis. A self-attentive model for knowledge tracing. *International Conference on Educational Data Mining*, 2019.

[69] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 1310–1318. PMLR, 2013.

[70] Richard J. Patz and Brian W. Junker. A straightforward approach to Markov Chain Monte Carlo methods for item response models. *Journal of Educational and Behavioral Statistics*, 24(2):146, 1999.

[71] Philip I. Pavlik, Hao Cen, and Kenneth R. Koedinger. Performance factors analysis –a new alternative to knowledge tracing. In *Conference on Artificial Intelligence in Education*, page 531–538, NLD, 2009. IOS Press.

[72] J Scott Payne and Jeff Allen. An early look at act academy usage and effectiveness. research report 2019-5. *ACT, Inc.*, 2019.

[73] Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J Guibas, and Jascha Sohl-Dickstein. Deep knowledge tracing. In *Advances in neural information processing systems*, pages 505–513, 2015.

[74] Elad Plaut. From principal subspaces to principal components with linear autoencoders. *arXiv:1804.10253v2*, 2018.

[75] Shi Pu, Geoffrey Converse, and Yuchi Huang. Deep performance factors analysis for knowledge tracing. In Ido Roll, Danielle McNamara, Sergey Sosnovsky, Rose Luckin, and Vania Dimitrova, editors, *International Conference on Artificial Intelligence in Education*, pages 331–341. Springer International Publishing, 2021.

[76] Hassan Rafique, Tong Wang, Quihang Lin, and Arshia Sighani. Transparency promotion with model-agnostic linear competitors. In *Proceedings of the International Conference on Machine Learning*, 2020.

[77] G. Rasch. Probabilistic models for some intelligence and attainment tests. *Danish Institute for Educational Research*, 1960.

[78] Mark D. Reckase. Multidimensional item response theory models. In *Multidimensional item response theory*, pages 79–112. Springer, 2009.

[79] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[80] Chet Robie, Michael J Zickar, and Mark J Schmit. Measurement equivalence between applicant and incumbent groups: An IRT analysis of personality scales. *Human Performance*, 14(2):187–207, 2001.

[81] Alexander Robitzsch, Thomas Kiefer, Ann Cathrice George, and Ali Uenlue. *CDM: Cognitive Diagnosis Modeling*, 2020. R package version 7.5-15.

[82] Raul Rojas. The backpropagation algorithm. In *Neural networks*, pages 149–182. Springer, 1996.

[83] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[84] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[85] Fumiko Samejima. Graded response model. In *Handbook of modern item response theory*, pages 85–100. Springer, 1997.

[86] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[87] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology*, 4(12):310–316, 2020.

[88] P Sibi, S Allwyn Jones, and P Siddarth. Analysis of different activation functions using back propagation neural networks. *Journal of theoretical and applied information technology*, 47(3):1264–1268, 2013.

[89] Sandip Sinharay and Russell G. Almond. Assessing fit of cognitive diagnostic models a case study. *Educational and Psychological Measurement*, 67(2):239–257, 2007.

[90] Karen L. Smarr and Autumn L. Keefer. Measures of depression and depressive symptoms: Beck depression inventory-II (BDI-II), center for epidemiologic studies depression scale (CES-D), geriatric depression scale (GDS), hospital anxiety and depression scale (HADS), and patient health questionnaire-9 (PHQ-9). *Arthritis Care & Research*, 63(S11):S454–S466, 2011.

[91] Kikumi K. Tatsuoka. Rule space: An approach for dealing with misconceptions based on item response theory. *Journal of Educational Measurement*, 20(4):345–354, 1983.

[92] Jonathan Templin and Lesa Hoffman. Obtaining diagnostic classification model estimates using mplus. *Educational Measurement: Issues and Practice*, 32(2):37–50, 2013.

[93] David Thissen and Howard Wainer. *Test Scoring*. Lawrence Erlbaum Associates Publishers, 2001.

[94] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[95] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning*, pages 1096–1103, 2008.

[96] Pauli Virtanen et al. Scipy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 2020.

[97] Zichao Wang, Yi Gu, Andrew Lan, and Richard Baraniuk. Varfa: A variational factor analysis framework for efficient Bayesian learning analytics. *arXiv preprint arXiv:2005.13107*, 2020.

[98] Mike Wu, Richard L Davis, Benjamin W Domingue, Chris Piech, and Noah Goodman. Variational item response theory: Fast, accurate, and expressive. *arXiv preprint arXiv:2002.00276*, 2020.

[99] Chun-Kit Yeung. Deep-IRT: Make deep learning based knowledge tracing explainable using item response theory. *CoRR*, abs/1904.11738, 2019.

[100] G.P. Zhang. Neural networks for classification: a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 30(4):451–462, 2000.

[101] Jiani Zhang, Xingjian Shi, Irwin King, and Dit Yan Yeung. Dynamic key-value memory networks for knowledge tracing. *26th International World Wide Web Conference, WWW 2017*, pages 765–774, 2017.

[102] Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. Interpretable convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8827–8836, 2018.

[103] Shengjia Zhao, Jiaming Song, and Stefano Ermon. Towards deeper understanding of variational autoencoding models. *arXiv preprint arXiv:1702.08658*, 2017.