DEVELOPING MACHINE LEARNING ALGORITHMS 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 Supervisor: Professor Suely Oliveira

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

PH.D. THESIS

This is to certify that the Ph.D. thesis of

Geoffrey Converse

has been approved by the Examining Committee for the thesis requirement for the Doctor of Philosophy degree in Applied Mathematical and Computational Sciences at the August 2021 graduation.

Thesis committee: _____
Suely Oliveira, Thesis Supervisor

_____
Mariana Curi

_____
David Stewart

_____
Jonathan Templin

_____
Colleen Mitchell

ACKNOWLEDGEMENTS

ABSTRACT

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 the learning of students, based on their performance on assessments. Item Response Theory (IRT) models the probabilty of students answering individual questions correctly as a function of the particular student's set of latent skills (e.g. add, subtract, multiply, divide), as well as parameters pertaining to the specific question, such as difficulty.

A common task is to measure a population of students' latent knowledge, given their responses to an assessment. Though many methods exist for this purpose, measurement becomes difficult as the number of students, items, and latent abilities increase. Specifically, when there are more than ten latent traits present, inferring student knowledge becomes infeasible. In this thesis, a novel neural network architecture is presented which estimates item and student parameters from high-dimensional assessment data with ease. This architecture is interesting from perspectives outside of education as well, as it introduces interpretability into an otherwise black-box machine learning model.

Due to the more recent emergence of online learning and AI tutoring systems, the application of knowledge tracing has gained more attention. In an online environment where many items are available, the goal is to track student's learning dynamically as they progress through an assessment. This information can be used

by intelligent tutoring systems to select which items to present to students, based on their personalized needs. In this thesis, modifications from the parameter estimation application are extended in order to integrate IRT into deep knowledge tracing models. The link between IRT and knowledge tracing provides additional model explainability, while remaining competitive with state-of-the-art methods.

The main contribution of this thesis is to present alternative methods for measuring student learning which thrive the the age of big data. This is helpful in the application area of education research and measurement – where traditional techniques face computational difficulties, the proposed machine learning driven approaches can efficiently handle high-dimensional data. From the computer science perspective, interpretability is inserted into otherwise unexplainable deep neural networks using domain-specific knowledge.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

Figure

CHAPTER 1
INTRODUCTION

TODO: I should have an introduction to the full work here.

1.1 Organization

This thesis is organized in three parts. In Part I, the first three chapters introduce Item Response Theory (IRT) and analyzes the novel parameter estimation method, ML2P-VAE. This method uses a modified variational autoencoder to estimation parameters in IRT models. Chapters 5-7 of Part II explore a task commonly seen in electronic learning environments in knowledge tracing. While other deep learning methods for knowledge tracing lack interpretability, new methods presented here present a trade-off between prediction power and explainability. Part III explores two applications of the ML2P-VAE method in areas outside of education: sports analytics and health sciences.

TODO: what is the best way to do this organization chapter?

CHAPTER 2
NEURAL NETWORKS AND EDUCATIONAL MEASUREMENT

TODO: more about why these two background sections are included

In this chapter, we present background information for Part I, parameter estimation in Item Response Theory, including basic neural network architectures, 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 [64], GPU technology has become more accessible and modern computers are more powerful, allowing anyone interested 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 [36] [77]. A brief overview of the mathematical inner workings of ANN is included in Appendix B.

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 someone's loan, the institution should be able to explain to the customer why they were denied [**?**]. 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 [50].

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 [33]. 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 [58]. 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 [79].

The field of education is an application 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 encoder input. 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 $\mathcal{A}(\boldsymbol{x}) := g(f(\boldsymbol{x})) : \mathbb{R}^n \to \mathbb{R}^n$. To train an AE, the loss function minimizes the difference between the input and output. This can be done in a number of ways, including the simple mean squared error loss

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

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 [56]. AEs have clear applications in image compression straight out-of-the-box, and can be modified for more complicated problems. Denoising autoencoders [74] 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.

could include citations: infoVAE, ELBO, "towards deeper understanding of VAE"

2.2 Variational Autoencoders

The motivation for designing a variational autoencoder (VAE), introduced by Kingma and Welling [41], is different from regular autoencoders in that it is not rooted in neural networks. Rather, a probabilistic point of view provides the main source of motivation, and neural networks are a very common tool used to implement a VAE.

Consider a dataset $\mathbb{X} = \{\boldsymbol{x}_i\}_{i=1}^N$, where each data point $\boldsymbol{x}_i \in \mathbb{R}^n$ 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 $\boldsymbol{x}_i$, a value $\boldsymbol{z}_i$ is first sampled from a prior distribution $p_*(\boldsymbol{z})$, and then x_i is generated from a distribution $p_*(\boldsymbol{x}|\boldsymbol{z})$.

From observing the dataset $\mathbb{X}$, the latent variables $\boldsymbol{z}_i$ and parameters of the distributions $p_*(\boldsymbol{z})$ and $p_*(\boldsymbol{x}|\boldsymbol{z})$ are unknown. The goal of a VAE is to approximate

these values, which leads to the ability to represent/encode data and generate new data. This should be done with a general algorithm that is unaffected by (i) intractability of the marginal likelihood $p(\boldsymbol{x}) = \int p(\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z})d\boldsymbol{z}$ and (ii) large amounts of data. Note that (i) is important because if $p(\boldsymbol{x})$ is intractable and thus $p(\boldsymbol{z}|\boldsymbol{x})$ is intractable, then the EM algorithm cannot be used.

maybe point out this is a common problem with IRT param est methods

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 $p_{\alpha^*}(\boldsymbol{z}|\boldsymbol{x})$ and $p_{\beta^*}(\boldsymbol{x}|\boldsymbol{z})$. In the encoder/decoder, the indices α and β reference settings of the trainable parameters of the neural network. Note that unlike a regular autoencoder, the probabilistic encoder $q_\alpha(\boldsymbol{z}|\boldsymbol{x})$ of a VAE outputs a 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 [8]. Consider a random variable x. For a particular value x_0, we can compute the amount of information gained from observing x_0 as $h(x_0) = -\log_2 p(x = x_0)$. When we use $\log_2$, the unit for information is "bits," but any logarithm can be used. Note that we gain more information from observing a low-probability event than from observing a high-probability event.

Entropy is defined as the expectation of $h(x)$: the average amount of information that will be learned by observing a random x. So the entropy of the 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 q instead of p. Cross-entropy is given as

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

We can simply define *Kullback-Leibler Divergence* (KL-Divergence) [42] as

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

Intuitively, KL-Divergence is amount of information which is lost if the approximating distribution $q(x)$ is used instead of the true distribution $p(x)$. However, KL-Divergence cannot be interpreted as a metric or as a distance between p and q because it is not symmetric – in general, $\mathcal{D}_{KL}[p||q] \neq \mathcal{D}_{KL}[q||p]$. KL-Divergence is non-negative, and $\mathcal{D}_{KL}[p||q] = 0$ iff $p(x) = q(x)$.

2.2.2 VAE Derivation

We derive the desired loss function for a VAE. The log marginal likelihood is given as

$$\log p_\beta(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) = \sum_{i=1}^{N} \log p_\beta(\boldsymbol{x}_i) \tag{2.6}$$

I don't know the best way to index p_β

Denoting $\boldsymbol{x} = \boldsymbol{x}_i$, we can rewrite each $p_\beta(\boldsymbol{x}_i)$ as

$$\begin{aligned}
\log p_\beta(\boldsymbol{x}) &= \int q_\alpha(\boldsymbol{z}|\boldsymbol{x}) \log p_\beta(\boldsymbol{x}) d\boldsymbol{z} \\
&= \int q_\alpha(\boldsymbol{z}|\boldsymbol{x}) \log \left(\frac{p_\beta(\boldsymbol{z}|\boldsymbol{x}) p_\beta(\boldsymbol{x})}{p_\beta(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} \right) \\
&= \int q_\alpha(\boldsymbol{z}|\boldsymbol{x}) \log \left(\frac{p_\beta(\boldsymbol{x}, \boldsymbol{z})}{p_\beta(\boldsymbol{z}|\boldsymbol{x})} \right) d\boldsymbol{z} \\
&= \int q_\alpha(\boldsymbol{z}|\boldsymbol{x}) \left(\log \frac{q_\alpha(\boldsymbol{z}|\boldsymbol{x})}{p_\beta(\boldsymbol{z}|\boldsymbol{x})} + \log \frac{p_\beta(\boldsymbol{x}, \boldsymbol{z})}{q_\alpha(\boldsymbol{z}|\boldsymbol{x})} \right) d\boldsymbol{z} \\
&= \mathcal{D}_{KL} \left[q_\alpha(\boldsymbol{z}|\boldsymbol{x}) || p_\beta(\boldsymbol{z}|\boldsymbol{x}) \right] + \int q_\alpha(\boldsymbol{z}|\boldsymbol{x}) \log \left(\frac{p_\beta(\boldsymbol{x}, \boldsymbol{z})}{q_\alpha(\boldsymbol{z}|\boldsymbol{x})} \right) d\boldsymbol{z} \\
&= \mathcal{D}_{KL} \left[q_\alpha(\boldsymbol{z}|\boldsymbol{x}) || p_\beta(\boldsymbol{z}|\boldsymbol{x}) \right] + \mathbb{E}_{q_\alpha(\boldsymbol{z}|\boldsymbol{x})} \left[-\log q_\alpha(\boldsymbol{z}|\boldsymbol{x}) + \log p_\beta(\boldsymbol{x}, \boldsymbol{z}) \right] \\
&= \mathcal{D}_{KL} \left[q_\alpha(\boldsymbol{z}|\boldsymbol{x}) || p_\beta(\boldsymbol{z}|\boldsymbol{x}) \right] + \tilde{\mathcal{L}}(\alpha, \beta; \boldsymbol{x})
\end{aligned} \tag{2.7}$$

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, and we can write

$$\log p_\beta(\boldsymbol{x}) \geq \tilde{\mathcal{L}}(\alpha, \beta; \boldsymbol{x}) = -\mathcal{D}_{KL} \left[q_\alpha(\boldsymbol{z}|\boldsymbol{x}) || p_\beta(\boldsymbol{z}) \right] + \mathbb{E}_{q_\alpha(\boldsymbol{z}|\boldsymbol{x})} \left[\log p_\beta(\boldsymbol{x}|\boldsymbol{z}) \right] \tag{2.8}$$

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 $\log p_\beta(\boldsymbol{x})$, even though we ignore the term $\mathcal{D}_{KL} \left[q_\alpha(\boldsymbol{z}|\boldsymbol{x}) || p_\beta(\boldsymbol{z}|\boldsymbol{x}) \right]$.

We take the ELBO $\tilde{\mathcal{L}}(\alpha, \beta; \boldsymbol{x})$ to be a potential VAE objective function which we want 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 $p_\beta(\boldsymbol{z})$. Note that unlike the true posterior $p_\beta(\boldsymbol{z}|\boldsymbol{x})$, the true prior is known and is nearly always assumed to be independent Gaussian.

For now, we assume that $\boldsymbol{z} \sim \mathcal{N}(0, I)$. Additionally, we assume that the encoder outputs a standard normal distribution. In Section 3.1.1, we propose architecture which fits the assumption that $\boldsymbol{z} \sim \mathcal{N}(\mu, \Sigma)$ 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 [28] that the KL-Divergence between an independent Gaussian distribution and a standard normal distribution of dimension K is calculated as

Should I show this?

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

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 is Bernoulli. To deal with 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 $L = 1$ [41]. So then

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

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 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 a neural network). Sample $\boldsymbol{z}^* \sim q_\alpha(\boldsymbol{z}|\boldsymbol{x}_0)$, and compute $\hat{\boldsymbol{x}} = p_\beta(\boldsymbol{x}|\boldsymbol{z}^*)$ (this is done by feeding $\boldsymbol{z}^*$ through a neural network). As demonstrated in Equation 2.9 and Equation 2.10, we can easily compute the objective function $\tilde{\mathcal{L}}(\alpha, \beta; \boldsymbol{x}_0)$.

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

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

where $\boldsymbol{z}^* \sim p_\beta(\boldsymbol{z}) = \mathcal{N}(0, I)$. 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 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, while regularizing on the latent code via the second term.

2.2.3 Implementation Details

Recall that each observed data point $\boldsymbol{x}_i \in \mathbb{R}^n$ and the corresponding latent code $\boldsymbol{z}_i \in \mathbb{R}^d$. 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 $p_\beta(\boldsymbol{z}) = \mathcal{N}(0, I)$.

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 at any point during training $\sigma_{0,k}^2 \leq 0$ 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 differentiable, 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 an optional

number of hidden layers to an output layer of size n. The output can be interpreted as a reconstruction $\hat{\boldsymbol{x}}_0$.

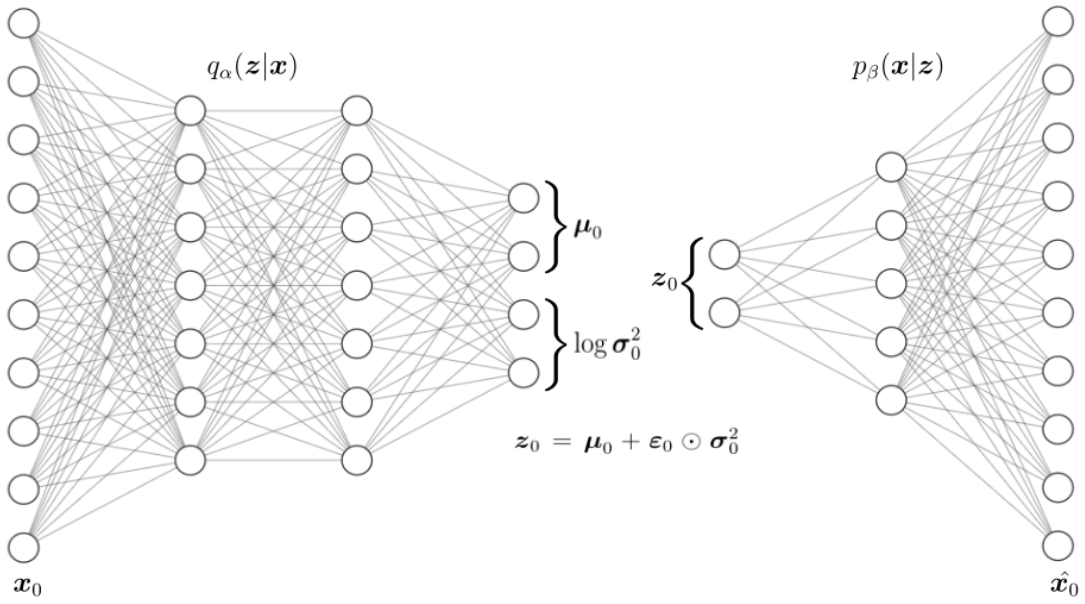

Figure 2.1: Visualization of a VAE architecture with $n = 10$ and $d = 2$.

The VAE architecture is summarized in Figure 2.1. 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) [72]. 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 wrong 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. True score theory 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 [69]. 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 [44]. 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, $\theta_j \sim \mathcal{N}(0, 1)$ [72].

In this work, we often consider the 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 $\mathbf{\Theta}_j = (\theta_{1j}, \theta_{2j}, \ldots, \theta_{Kj})^\top$. It is then assumed that the latent abilities of students follow some multivariate Gaussian distirbution, $\mathcal{N}(0, \Sigma)$. 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. Instead, the usual task is to infer student's knowledge $\boldsymbol{\Theta}_j$ from their responses to some assessment containing n questions, referred to as items. A student'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 2.2.

It is worth mentioning that IRT models also exist for non-binary response data. For example, Samejima's graded response model [66] allows for items to be answered in multiple categories or grades. This idea is related to partial credit scoring [45], 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

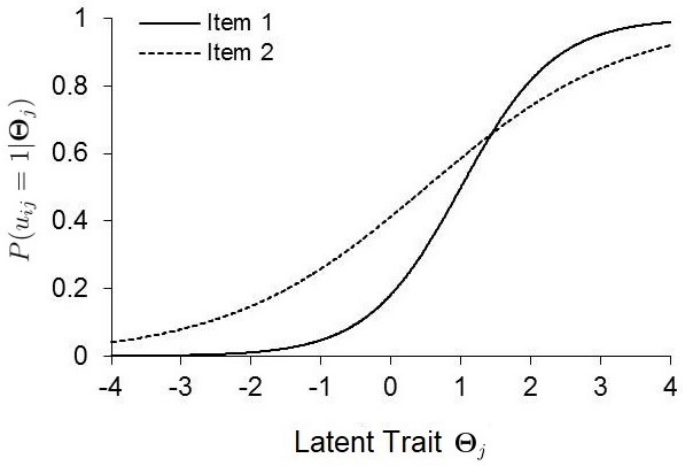

Figure 2.2: An item characteristic curve visualizes the relation between a student's ability and the probability of answering an item correctly.

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 Θ_j is a vector.

2.3.1 Rasch Model

One of the first IRT models was proposed by Georg Rasch in 1960 [59]. 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 $\frac{\xi}{\xi+\delta} = \frac{1}{1+\delta/\xi}$ and note that $\frac{\xi}{\xi+\delta} \to 1$ as $\xi \to \infty$. After the reparametarization $\xi = e^{\theta}$ and $\delta = e^{b}$, we arrive at the 1-Parameter Logistic Model, often referred to as the Rasch Model [72].

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

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 2.2, 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 [72]. 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 2.2 – 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 give 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 [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 $\boldsymbol{\Theta}_j = (\theta_{j1}, \ldots, \theta_{jK})^\top$. 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.

TODO: mention MDISC and how this scales

A few summary statistics for items modeled by MIRT are often of interest [60]. 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 [70] $Q \in \{0,1\}^{n \times K}$ 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 need 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. [24]:

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

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 cor-

rect/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

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

In the folowing 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 [48]. 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), the system is solved iteratively via Newton-Raphson methods [5].

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

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, $\frac{\partial^2 \log L}{\partial\theta_{jk}\partial\theta_{j'l}} = 0$ 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': $\frac{\partial^2 \log L}{\partial b_i \partial b_{i'}} = 0$ for $i \neq i'$ (the same applies to discrimination parameters a_{ik}). Now, B is in 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: $\frac{\partial^2 \log L}{\partial \theta_{jk} \partial b_i} := 0$ for all j, k, i.

The logic behind this assumption is that "there is no reason to believe that there should be any covariaion between an individual eaminee 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

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

Note that $u_{ij} \in \{0, 1\}$, giving two simpler cases for $\frac{\partial^2 \log L}{\partial \theta_{jk} \partial a_{ik}}$. There exist many settings of a_{ik} and θ_{jk} such that $\frac{\partial^2 \log L}{\partial \theta_{jk} \partial a_{ik}} \neq 0$. 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 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 computatationally 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. Even if given those resources, as the number of latent traits K grows, the individual sub-matrices become more and more difficult to invert.

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 get wrong is usually interpreted as a rather difficult item. But JMLE takes this

to the extreme, and the difficulty parameter is unboundedly increased 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:

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

I think my math is wrong: this may not be cc up?

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 [65].

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 intial 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

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

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 [7] for JMLE for Newton-Raphson iterations in Equation 2.26. The two simplifications of constraining 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 distrubtion 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 $\frac{d}{dx}g(x) = g(x) \cdot \left[\frac{d}{dx}\log g(x)\right]$ for all continuously differentiable $g : \mathbb{R} \to (0, \infty)$ 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 [26] [10]. Some modifications to the EM algorithm use Monte Carlo methods to compute the integral (MCEM) [46]. The K-dimensional integral is discretized into $H = c_1 \times c_2 \times \cdot \times c_K$ 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 perform 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 $H = 3^{10} = 59,000$ 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 [38] with the Robbins-Monro stochastic optimization algorithm [61] 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 [37].

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.

TODO: add more info on MHRM at some point

When trying to implement MHRM 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 $J(\boldsymbol{x}) = \frac{\partial^2 \log L}{\partial x_i \partial x_j}$ 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.

TODO: What
TODO: I *think*
if I move
this is why
neural net
there is still an
background,
issue..
then put

Variational

IRT and others

in this section?

CHAPTER 3
ESTIMATING IRT PARAMETERS WITH VARIATIONAL AUTOENCODERS

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 can produce estimates as accurate as those of traditional parameter estimation techniques. Further, ML2P-VAE scales much better than traditional methods as the number of latent abilities becomes large. In the field of machine learning, ML2P-VAE is an unsupervised learning technique which yields explainable results. A variational autoencoder is used in an unorthodox way, and the trainable parameters and a hidden neural layer are able to be understood in a real-world context.

3.1 ML2P-VAE Method Description

Assume we are given an assessment with n items which tests K latent skills, and that N students take this exam. Data is given as an $N \times n$ binary matrix, as in Equation 2.12. No information about student latent ability parameters $\Theta \in \mathbb{R}^K$ or item parameters a_{ik} and b_i is provided. However, there is access to an expert-annotated binary Q-matrix detailing the item-skill associations, as in Equation 2.21.

TODO: relate to IRT background section and VAE derivation

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 representing one item. The inputs are be the binary response vectors $\boldsymbol{u}_j$ and the outputs are approximations of the probability of student j answering each item correctly. Next, 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 modifications to a typical VAE architecture are focused on the decoder. No hidden layers are used in the decoder. Instead, a non-dense set of weights connects the decoder input to the decoder output. The non-zero weights here are determined by the Q-matrix; 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.

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

$$\sigma(z_i) = \frac{1}{1+e^{z_i}} \tag{3.1}$$

in the output layer. Here, $z_i = \sum_{k=1^K} w_{ik}\alpha_k + \beta_i$, where w_{ik} is the weight between the k-th and i-th nodes in the decoder input and decoder output layer, α_k is the activation of the k-th node in the decoder input layer, and β_i is the additive bias in the output layer. Note the similarity between Equation 3.1 and Equation 2.18. The constraint on the weights along with the sigmoid activation function allows for interpretation of the decoder as an ML2P model.

Specifically, the 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 response input $\boldsymbol{u}_j$) can be interpreted as estimates to the student ability parameter θ_{kj}.

may want to mention that the value outputted by encoder is then sampled from for decoder input

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.

explain what this means $-\theta a_{ik} = \theta(-a_{ik})$

Relating the ML2P-VAE model back to the derivation of a variational autoencoder in Section 2.2.2, the notations used are related as follows. The response vectors $\boldsymbol{u}_j$ serve as the observed data $\boldsymbol{x}$, and the latent ability of students $\boldsymbol{\Theta}_j$ serve as the latent code $\boldsymbol{z}$. The modified decoder, which produced outputs the probabilities of student j answering item i correctly $\hat{P}_{ij} = \sigma(\theta_{ij})$, as described in Equation 3.1, serves as the observed input reconstruction $\hat{x}_i$ in Equation 2.10.

3.1.1 Full Covariance Matrix Implementation

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 the latent prior $p(\Theta)$ 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 is often beneficial to force the latent dimensions to be independent of one another.

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, and

(2) calculating Kullback-Leibler Divergence

These two characteristics must be addressed when constructing the neural architecture.

After training a VAE, sending a data point 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 $\mu_0 \in \mathbb{R}^K$ and a symmetric, positive-definite matrix $\Sigma_0 \in \mathbb{R}^{K\times K}$. Sampling from $\mathcal{N}(\mu_0, \Sigma_0)$ requires a matrix G_0 such that $G_0 G_0^\top = \Sigma_0$. This matrix factorization G_0 is not unique, but it can be convenient to use the Cholesky decomposition of Σ_0 [3]. The sample of the multivariate Gaussian is calculated as $z_0 = \mu_0 + G_0\boldsymbol{\varepsilon_0}$, 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

$$\begin{aligned}&\mathcal{D}_{KL}\left[\mathcal{N}(\mu_0, \Sigma_0)||\mathcal{N}(\mu_1, \Sigma_1)\right] = \\ &\frac{1}{2}\left(\mathrm{tr}(\Sigma_1^{-1})\Sigma_0) + (\mu_1 - \mu_0)\Sigma_1^{-1}(\mu_1 - \mu_0) - K + \ln\left(\frac{\det\Sigma_1}{\det}\Sigma_0\right)\right)\end{aligned} \tag{3.2}$$

When using this in a VAE, $\mathcal{N}(\mu_1, \Sigma_1)$ corresponds to the prior $p(\Theta)$, and so μ_1 and Σ_1 are constant. Then 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 3.2 computes $\ln\det\Sigma_0$, so we must have $\det\Sigma_0 > 0$ at any point during training. Recall that μ_0 and Σ_0 correspond to the input 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 u_0 at any point during training, then it is not possible to compute the loss and gradient. Thus, a specific architecture which guarantees that $\det\Sigma_0 > 0$, regardless of the input 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. After a sufficient number of hidden layers of sufficient size, the encoder outputs $K + \frac{K(K+1)}{2}$ nodes. The first K nodes represent the mean vector μ_0, and the remaining $\frac{K(K+1)}{2}$ nodes are arranged into a lower triangular matrix $L_0 \in \mathbb{R}^{K\times K}$. The covariance matrix is obtained by using the matrix exponential $\Sigma_0 = e^{L_0} \cdot \left(e^{L_0}\right)^\top$.

Theorem 3.1. Σ_0 *constructed as described previously 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, $\det \Sigma_0 = \det G_0 \cdot \det G_0^\top > 0$. So now for any nonzero $x \in \mathbb{R}^K$,

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

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 u_0 to a multivariate Gaussian distribution $\mathcal{N}(\mu_0, \Sigma_0)$. Additionally, the sampling operation and KL-Divergence calculation can always be carried out without issue. A visualization of the ML2P-VAE architecture for correlated latent traits is shown in Figure 3.1.

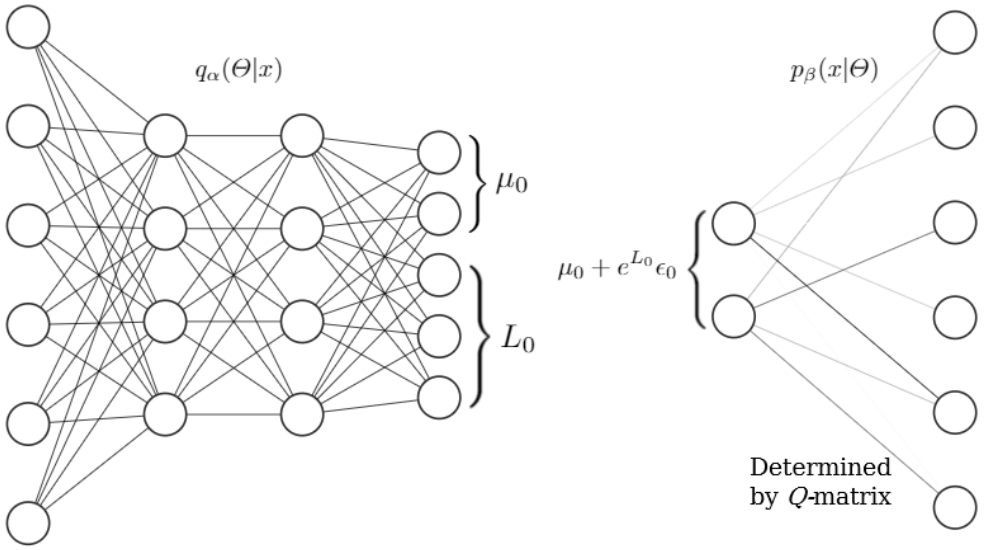

Figure 3.1: Visualization of the ML2P-VAE architecture for two correlated latent traits and six input items. Note that the trainable weights matrix in the decoder is not dense, but is determined by the given Q-matrix.

3.1.2 Variants of ML2P-VAE

We consider three scenarios for using ML2P-VAE in practice: (a) the best case scenario where we assume that the covariance matrix between all latent traits is known, (b) we don't know the exact covariance matrix, 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 scenario (b), we multiply the response matrix (N students by n items) by the Q-matrix (n items by K abilities). We then take the Pearson correlation of the columns of this $N \times K$ matrix to obtain an approximate correlation matrix in $\mathbb{R}^{K \times K}$ between abilities.

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

$$M = U \cdot Q, \quad M \in \mathbb{R}^{N \times K}$$

$$\hat{\Sigma}_{kl} = \frac{\sum_{i=1}^{N}(k_i - \bar{k})(l_i - \bar{l})}{\sqrt{\sum_{i=1}^{N}(k_i - \bar{k})^2}\sqrt{\sum_{i=1}^{N}(l_i - \bar{l})^2}} \tag{3.3}$$

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 as in Equation 2.15, rather than the ML2P model as in 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. This still allows for interpretation of the learned distribution of the VAE as estimates to student abilities Θ, 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 [18]. 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'')
```

maybe use lstinline from listings for code

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 5 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 1-Parameter Logistic model, also called the Rasch model. 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.

Should i include a short code demo?

CHAPTER 4
ML2P-VAE RESULTS AND DISCUSSION

The ML2P-VAE method has been used in a various settings in multiple publications [23, 17, 20]. The first paper introduced the method and gives some preliminary results on a small simulated data set. The second, a follow-up presented at the Conference for Artificial Intelligence in Education, displays the advantages that a VAE holds over a regular autoencoder in the task of parameter estimation. The final publication, which has been submitted to Machine Learning, compares different variations of ML2P-VAE with traditional parameter estimation methods on both real and simulated data sets of various sizes.

4.1 Description of Data Sets

Sim-ECPE

This simulated data set is designed to mirror the real-life Examination for the Certificate of Proficiency in English, detailed further in the next 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. Up to 10,000 student abilities $\Theta \in \mathbb{R}^3$ were sampled from $\mathcal{N}(0, I)$. Note that in Sim-ECPE, it is assumed that the latent traits are independent. We use a Q-matrix consistent with previous literature [24, 71, 39].

Define MDISC

ECPE

The Examination for the Certificate of Proficiency in English (ECPE) is an exam with 28 items. The set of responses we use is available in the **CDM** package for R [62]. 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.

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_{ki} is sampled from Bern(0.2). To ensure each item requires at least one latent ability, if a column $q_{:i} = 0$ after sampling, then one random element in the column is changed to a 1. The discrimination parameters are chosen so that $a_{ik} \in [0.1, 1.3]$ and $b_i \in [-3, 3]$. Abilities $\Theta \in \mathbb{R}^6$ 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 [75].

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 exact same was as that of Sim-6, where $q_{ki} \sim$ 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, and the correlation matrix is generated in the same manner.

Sim-4

Sim-4 contains 3,000 student's 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

TODO: edit this section

This section describes the experiments performed in [23], when the ML2P-VAE model was initially proposed.

Data is simulated from the ML2P model with a Q-matrix to determine the relationship between the items and the latent traits. A scenario with $n = 28$ items evaluating $K = 3$ latent traits is considered. The data is designed to mirror a real assessment already adopted by other authors, based on the Examination for the

TODO: define MDIFF and MDISC somewhere before this

Certificate of Proficiency in English (ECPE) for nonnative English speakers. The same Q-matrix studied in prior IRT literature is used [12, 39, 71, 24]. The values of discrimination and difficulty parameters were generated from a uniform distribution such that $a \in (0.25, 1.75)$ and $b \in (-3, 3)$. The the discrimination parameters were chosen such that $0.25 \leq MDISC_i \leq 1.75$, for all i, to obtain feasible values of the item discrimination parameters.

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 the 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. [24], 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.

Given a subject's latent traits, we generate ten different replicates of responses using the M2PL-Q method [24] (considering the ML2P model and Q-matrix). 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 $\hat{a}$, $\hat{b}$, and $\hat{\Theta}$.

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.

As mentioned previously, 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 M2PL-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 used to generate the data. So although 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.

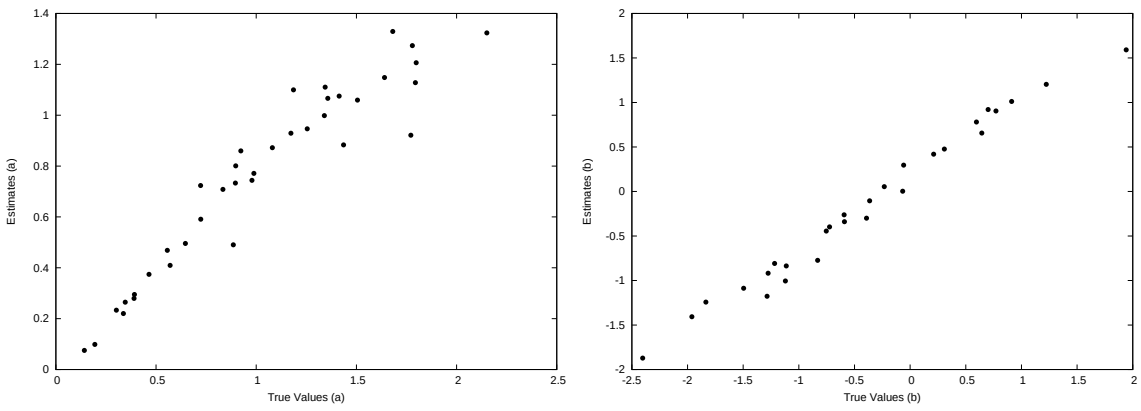

Figure 4.1: True versus estimated values for discrimination parameters (left) and difficulty parameters (right) with sample size 10,000.

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 model. The estimates for a can be used to quantify 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 $\hat{b}$ 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.

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

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

where λ_i refers to one of the parameters of the model (a_{ik}, b_i, or θ_{jk}) and $\hat{\lambda}_{ip}$ refers to the respective estimate for data replicate p. 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 a estimates decrease, as is expected. Oddly, the error measures for b increase as the dataset increases - this issue is left for future work. However, this does not seem to affect the correlation between true and estimated b.

The encoder of our VAE also holds predictive power. Given a subject's as-

AVRB				
Size	a_1	a_2	a_3	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	a_1	a_2	a_3	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

the b values are likely wrong

CORR				
Size	a_1	a_2	a_3	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: Statistics for parameter estimates.

sessment results, we feed this information forward through the encoder and return a prediction of that subject's latent traits. This is due to using the Q-matrix in determining the nonzero weights of the decoder and allows us to interpret what the distribution in the variational autoencoder is learning, a rather unique characteristic among typical variational autoencoders. Usually, this distribution is very abstract and non-interpretable, but the Q-matrix allows us to directly relate this distribution to our subjects' latent traits.

In Figures 4.2, 4.3, and 4.4, we observe an explicit relationship between the learned distributions $\hat{\theta}_i$ of the VAE and the latent traits θ_i. The plots seem to have a sigmoidal tendency, rather than linear. This is not ideal, as it causes our model to

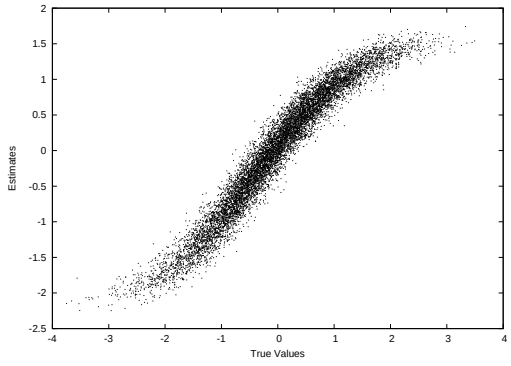

Figure 4.2: $\hat{\theta}_1$ estimates for the first latent variable.

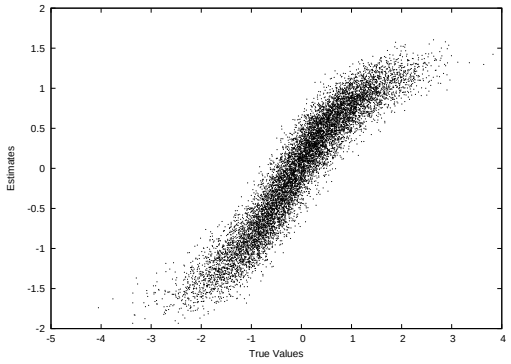

Figure 4.3: $\hat{\theta}_2$ estimates for the second latent variable.

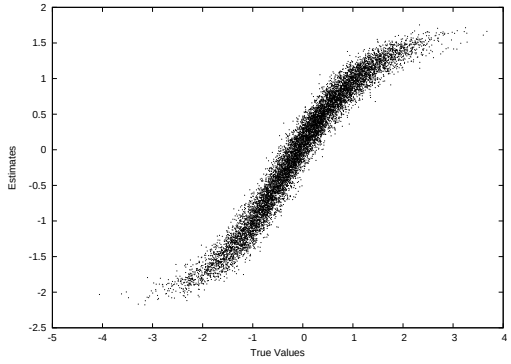

Figure 4.4: $\hat{\theta}_3$ estimates for the third latent variable.

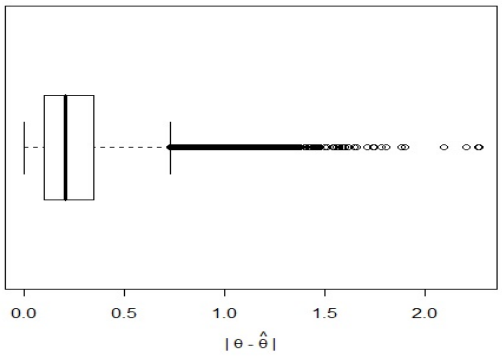

Figure 4.5: Absolute differences between true values and estimates of θ.

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 under 0.5, which improves upon the results of the traditional estimation methods in MIRT [24], which achieved approximately 75% of the individuals with absolute bias

under 0.5.

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.

The latent traits 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 show the relation between the most used multidimensional IRT model and VAE. Using the VAE formulation to estimate MIRT parameters solves one important limitation of the traditional estimation processes: MCMC and MML. Until now, the point and standard error estimation of latent trait parameters from MIRT models with medium to large dimension was infeasible.

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 MML) give better results. For example, da Silva et al. [24] find good estimates with smaller sample sizes for the same experimental situation using MCMC, but with a running time around 40× longer for $N = 500$, and 108× 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. It is an issue that should be considered for future studies. This conclusion is supported by scientific reseachers in statistics [9].

4.2.2 Variational Autoencoder vs Autoencoder

Shortly after introducing the ML2P-VAE method, comparisons between a variational autoencoder (VAE) and a regular autoencoder (AE) for parameter estimation were made [17]. Recall that Guo et al. proposed a neural network approach to estimating student mastery in 2017 [34]. This neural network had autoencoding structure, but was geared towards CDM and did not make a connection to IRT or parameter estimation. In this section, we show empirically that using a VAE produces better item and ability estimates than a regular autoencoder and analyze the differences in models leading to this imporvement.

For these experiments, the same simulated data presented in Section 4.2.1 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 so that the VAE encoder produces a standard normal distribution.

Model	a_1	a_2	a_3	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: Statistics for item parameter recovery.

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 a_k denotes the average measure taken over all items related to latent trait θ_k, and b is the average measure taken over all item difficulty 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. 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.

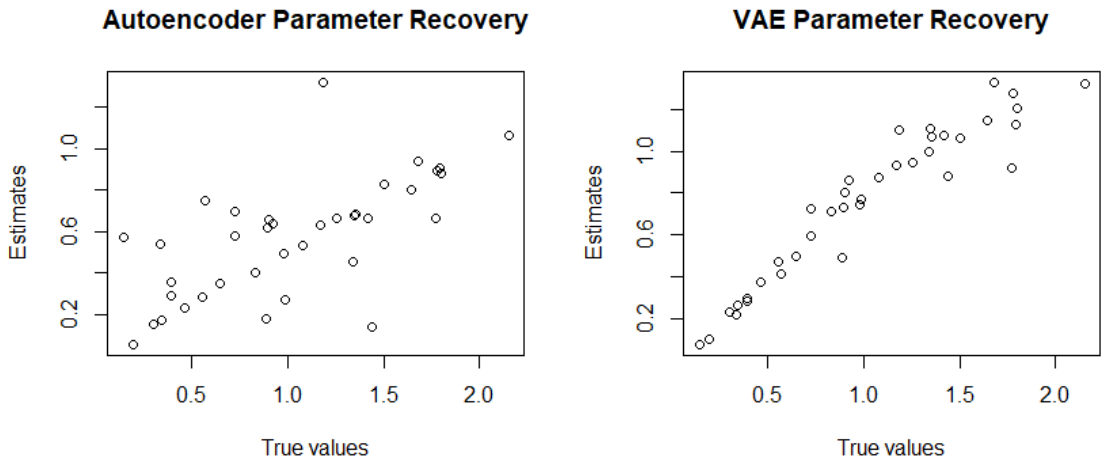

Figure 4.6: Autoencoder and VAE discrimination parameter (a_{ji}) recovery.

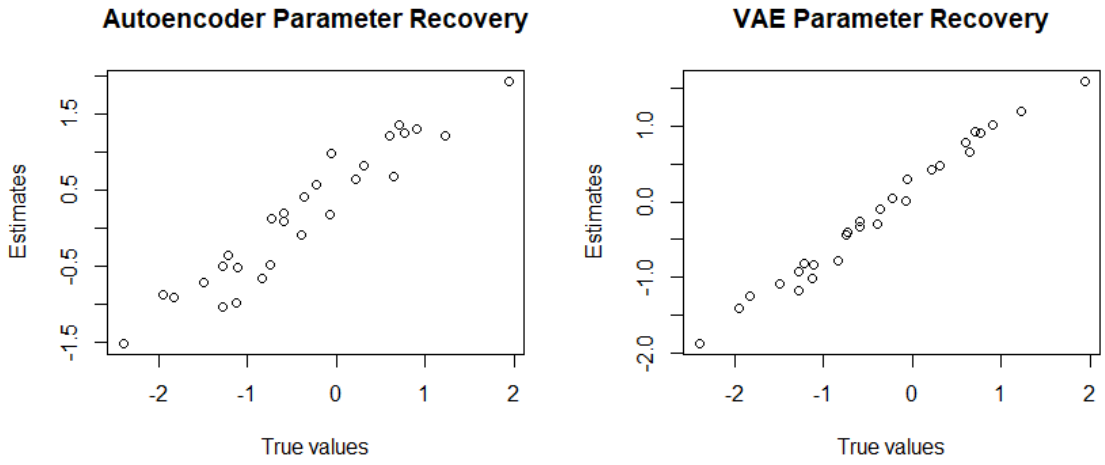

Figure 4.7: Autoencoder and VAE difficulty parameter (b_i) recovery.

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 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 error measures yes good correlation 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 is on a different scalethan that of the right plot. This is likely due to the fact that a VAE has a KL-divergence term in its loss function.

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: Statistics for latent trait prediction.

The lack of a KL-divergence term in an AE also helps explain the poor discrimination parameter estimates shown in the right plot of Figure 4.6. The ML2P model can suffer from an identifiability issue without the assumption that student ability parameters follow some probability distribution [35]. Adding a KL-divergence term in the VAE loss function between the encoder output and the prior $p(\theta)$, which is $\mathcal{N}(0, I)$ in this case.

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

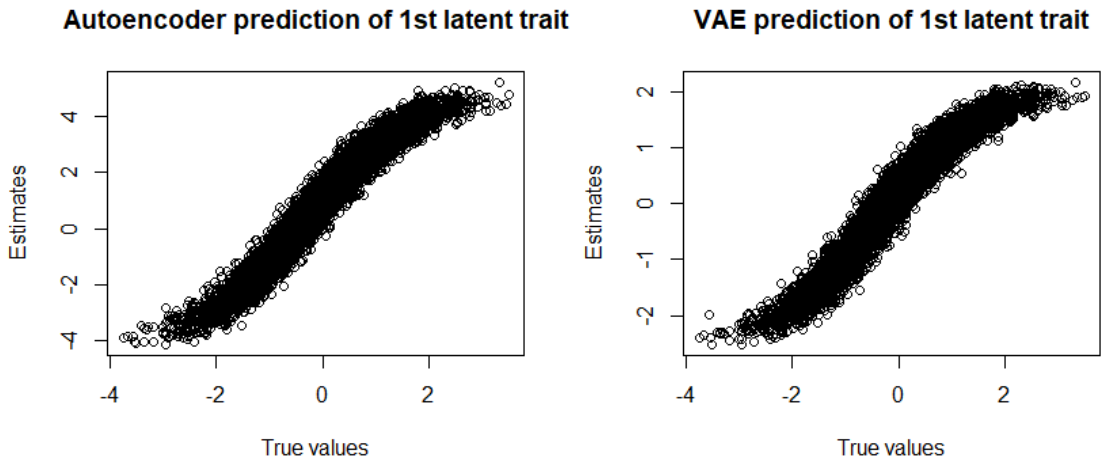

Figure 4.8: Autoencoder and VAE predictions for θ_1.

incorporate an extra piece of domain knowledge in the prior distribution of Θ, leading to more accurate estimates.

4.2.3 ML2P-VAE vs Traditional Methods

TODO: do 1pl in mirt and VAE

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) [10]. This work has been submitted to the Machine Learning journal [20].

update if this ever gets accepted

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. 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 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. While quadrature or MCMC methods become infeasible on data sets much 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

Data Set	Method	a.RMSE	a.BIAS	a.COR	b.RMSE	b.BIAS	b.COR	θ.RMSE	θ.BIAS	θ.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	192s
	MCEM	0.0228	0.0148	0.9998	0.0064	-0.0008	1	0.0132	0.0026	0.9998	33s
	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

Table 4.4: Error measures for discrimination (a), difficulty (b), and ability (θ) 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

dimension 6-20), the longer runtime is more due to the number of data samples, rather than the size of the latent dimension. 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 used for image classification.

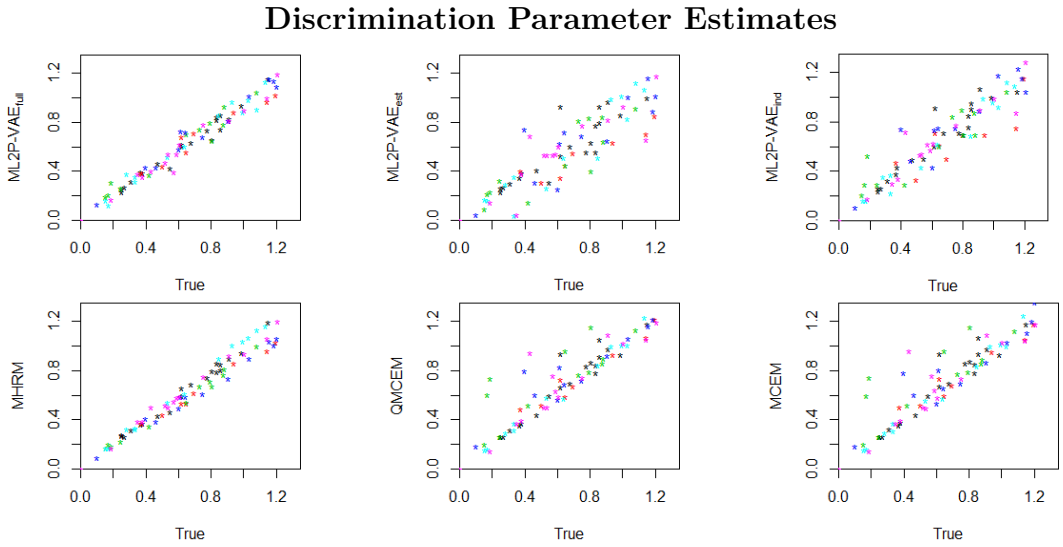

Figure 4.9: Correlation plots of discrimination parameter estimates for the Sim-6 dataset with 50 items and 6 latent traits. ML2P-VAE estimates are on the top row, and traditional method estimates are on the bottom row

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 latent ability associated with the ability or discrimination parameter. 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 difficultly

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 do not obtain the same levels of accuracy as when assuming full knowledge of the latent ability correlations.

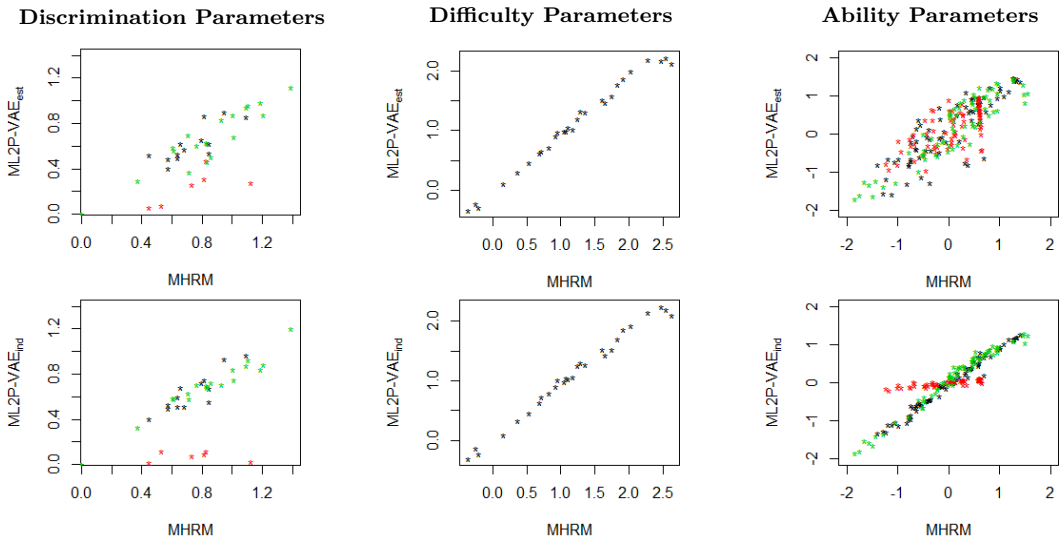

Figure 4.10: Estimates from ML2P-VAE methods plotted against "accepted" MHRM estimates from the ECPE dataset

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.

While estimating parameters for the Sim-20 dataset, the dimension of the latent traits ($\mathbb{R}^{20}$) is too large for traditional methods, 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. The discrimination parameter estimates for ML2P-VAE$_{est}$ seem to have no pattern. Upon closer inspection, it can be seen that the discrimination parameter estimates associated with a particular ability are correlated, but each ability is on a different scale.

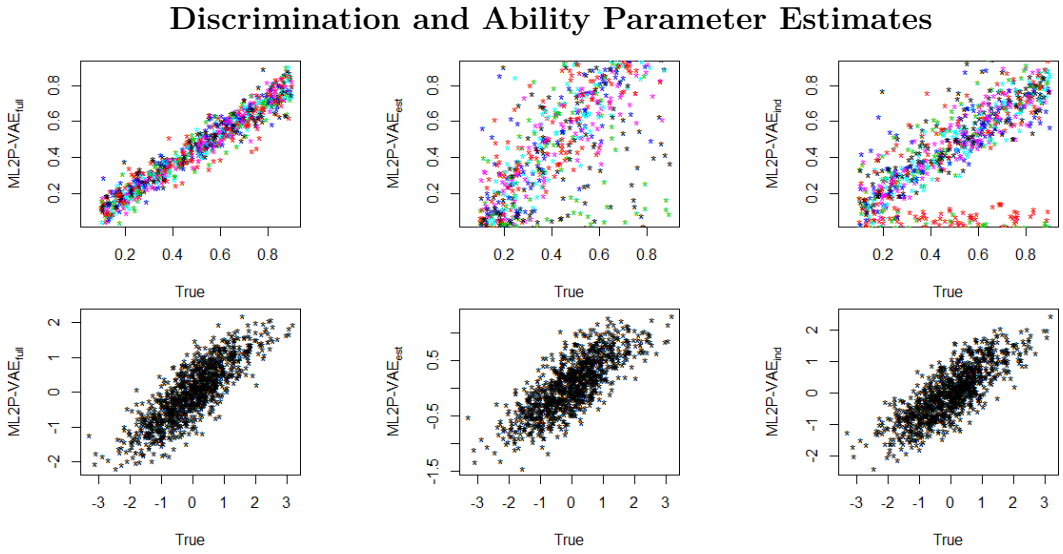

Figure 4.11: ML2P-VAE parameter estimates for Sim-20 with 200 items and 20 latent traits. The top row shows discrimination parameter correlation, and the bottom row shows ability parameter correlation

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 in the correlation matrix estimation 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 θ 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.

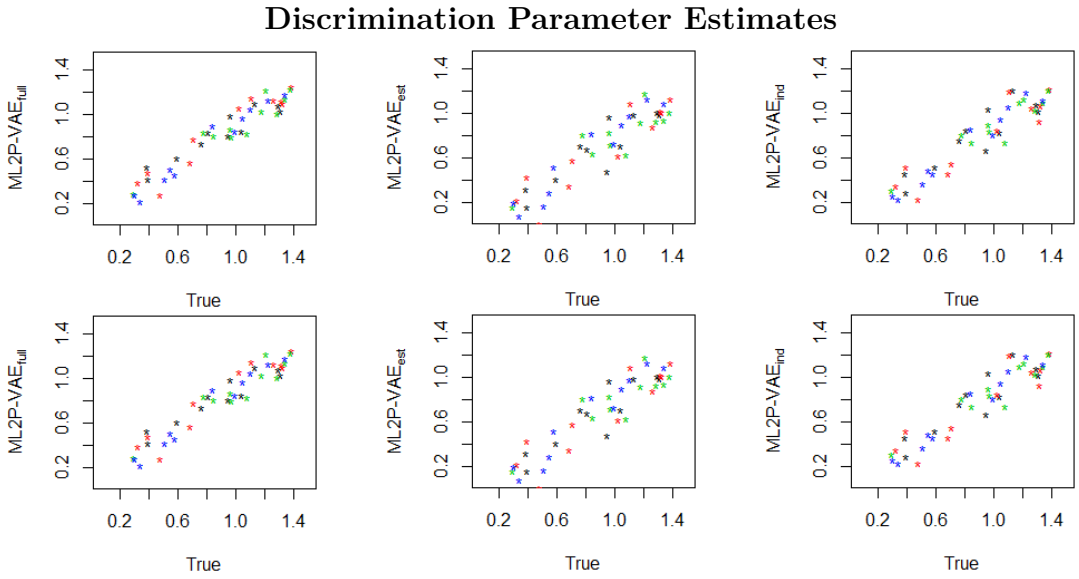

Figure 4.12: Discrimination parameter estimates for Sim-4 with 27 items and 4 latent skills. The top row shows estimates from ML2P-VAE methods, and the bottom row gives estimates yielded by traditional methods.

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 ability θ and item difficulty 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-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.

The longer runtimes in Table 4.4 for Sim-20 can be attributed more to the fact

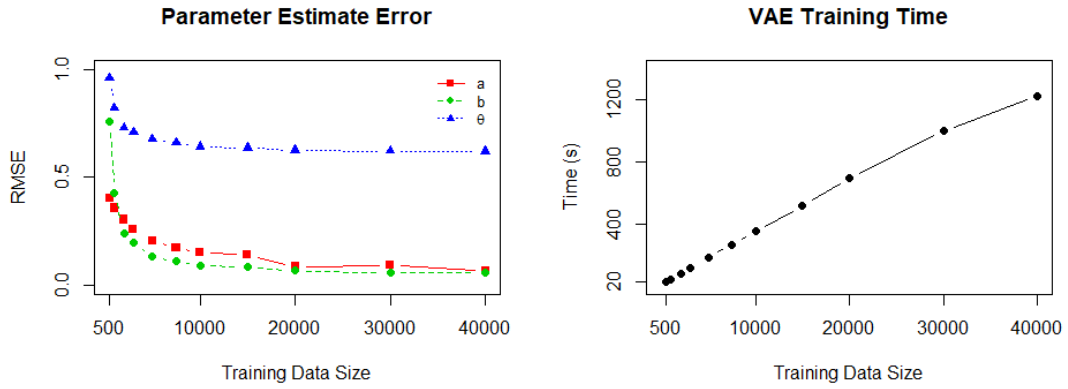

Figure 4.13: Performance of ML2P-VAE$_{full}$ on data set (iii) when trained on data sets of increasing size. The left plot gives the test RMSE after using different sizes of training data, and the right plot shows the time required to train the neural network

that there were 50,000 data samples, 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.

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 Θ, whereas traditional methods are unable to estimate high-dimensional Θ. Estimating the difficulty parameters b is manageable with a smaller data set, while discrimination parameters require a large amount of training data to obtain quality

estimates.

am I going to do anything with 3PL?

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. 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 [7], 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.

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. While this technique 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 Θ_j are not directly trainable parameters of the neural network – the partial derivatives $\frac{\partial \mathcal{L}}{\partial \theta_{jk}}$ are never calculated or directly optimized.

Of course, the most accurate ML2P-VAE method 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. Decreasing this hyper-parameter gives more emphasis on reconstructing inputs, rather than fitting data to 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 esti-

mates Θ_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
TIME-SERIES NEURAL NETWORKS AND KNOWLEDGE TRACING

TODO:
TODO: is
explain why I
"Time-Series"
have
best to use?
background on
Maybe
TSNN and KT
"Temporal"?
here

Time-Series Neural Networks

In many deep learning applications such as video processing, natural language processing, or dynamical systems, the observed data is time-dependent [29] [73] [32].In such datasets, a single observation of d features can not represented as a vector $\boldsymbol{x}_0 \in \mathbb{R}^d$, but must take into account the T different measurements of the d features, each taken at a different timestep $1 \leq t \leq T$. As such, a data point is represented as a matrix $X_0 \in \mathbb{R}^{d \times T}$, where each column t of X_0 gives a snapshot of the observation at time t.

5.1 Recurrent Neural Networks

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

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

where $W \in \mathbb{R}^{h \times d}$ and $\boldsymbol{b} \in \mathbb{R}^h$ are trainable parameters and f is a non-decreasing activation function [67].

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 $W_{hh} \in \mathbb{R}^{h\times h}$, $W_{hx} \in \mathbb{R}^{h\times d}$, $\boldsymbol{b}_h \in \mathbb{R}^h$, $W_{hy} \in \mathbb{R}^{h\times(h+d)}$, and $\boldsymbol{b}_y \in \mathbb{R}^h$ are trainable parameters [30]. The notation $[\boldsymbol{x}_t, \boldsymbol{h}_t]$ refers to vector concatenation. Note that this allows the output $\boldsymbol{y}_t$ to include information from previous time-steps. A visualization of an unfolded RNN is shown in Figure 5.1.

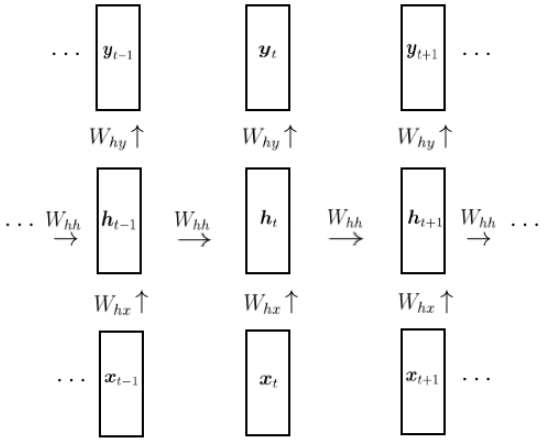

Figure 5.1: Architecture of a recurrent neural network.

One issue that RNN face is the exploding or vanishing gradient problem [6], 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 [53]. 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 time-steps 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 this issue, Long Short-Term Memory (LSTM) networks were developed by Hochreiter and Schmidhuber [40]. This architecture introduces element-wise multiplication and addition operations in addition to 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 time-step 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 [40].

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, given an input t $\boldsymbol{x}_t$, 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 near zero) of $\boldsymbol{c}_{t-1}$ are forgotten:

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

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

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

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 time-step:

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

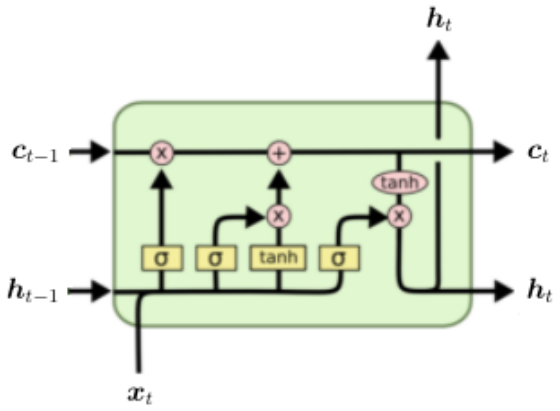

Figure 5.2: Architecture of a single LSTM cell [51]. Trainable matrix multiplication followed by an activation function are in yellow boxes, and element-wise operations without learned parameters are in red ovals.

The architecture of an LSTM 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 [73].

While transformers are large neural networks with many components and parameters, they lean heavily on the attention mechanism. Given a d-dimensional feature vector $\boldsymbol{x}_t$ at each time-step $1 \leq t \leq T$, define three trainable matrices W_q, W_k, and W_v. These are used to obtain a *query*, *key*, and *value* vectors $\boldsymbol{q}_t$, $\boldsymbol{k}_t$, and $\boldsymbol{v}_t$ for each time-step. Note that these can be arranged into matrices $Q, K, V \in \mathbb{R}^{T \times d}$. Calculated the correlation between observation t and all other time-steps:

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

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 $\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{T} e^{z_j}}$ 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 $i > t$.

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 time-step, weighted by the correlation scores in Equation 5.7. The attention calculation can also be written more generally:

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

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" [73]. These blocks are stacked on top of each other up to six times to obtain deeper and deeper contextualization of the input sequence [25]. 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 developed 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 a d-dimensional learned representation of an individual word [47]. Then the correlation score in Equation 5.7 is quantifying the relationship between pairs of words in the sentence.

Knowledge Tracing

Knowledge Tracing (KT) is a task introduced by Corbett and Anderson in 1995 [22]. 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 n is

$$P(L_n) = P(L_{n-1}|\text{evidence}) + (1 - P(L_{n-1}|\text{evidence})) \cdot P(T). \tag{5.10}$$

Then the probability of a student performing a task correctly is the sum of the probability that the rule is learned and the student doesn't make a mistake, and the probability that the rule is unlearned but the student guesses correctly.

There are only four parameters for each rule: the probability that the rule is already in the learned state at timestep 0, the probability of transitioning from the unlearned to learned state, the probability of guessing correctly, and the probability of slipping. These parameters are estimated using a hidden Markov Model, and the probability of a student having learned a rule is updated via Bayes' Theorem.

In recent years, Bayesian Knowledge Tracing (BKT) has been overcome by deep learning methods. The popularity of neural networks has brought black-box models that yield high accuracy. Many of these methods, detailed in Section 5.3, 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.

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 hyper-parameter 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 160 questions, then this student's interactions will be split into two separate sequences of length 128 and 32.

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 correctly or not. So for learning system with n available questions, there are $2n$ possible interactions for x_t. The knowledge tracing task is to predict c_{t+1} given all previous interactions. Mathematically, the quantity of interest is the probability

$$P(c_{t+1} = 1 | (q_0, c_0), (q_1, c_1), \ldots, (q_t, c_t), (q_{t+1}, ?)). \tag{5.11}$$

Most neural networks optimize the predicted probability in Equation 5.11 by minimizing the cross-entropy loss function, as described in Equation 2.2.

5.4 Deep Knowledge Tracing

In 2015, the first use of neural networks for knowledge tracing was introduced by Piech et al. [55]. 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. RNN

are the most simple neural network to deal with sequential time-series data. LSTM are more sophisticated, and are capable of capturing longer-range dependencies due to their "keep/forget" functionality.

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 of 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, the d-dimensional embedding, 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 that 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.

5.5 Dynamic Key-Value Memory Networks

More sophisticated neural network approaches to knowledge tracing were introduced by Zhang et al. with Dynamic Key-Value Memory Networks (DKVMN) [78]. They modify a memory-augmented neural network (MANN) in order to fit into the knowledge tracing framework. A MANN is a time-series neural network, but it

does not rely on residual connections like an RNN or LSTM. Rather, a value matrix M^v is stored in memory for each student, and the entries in M^v are updated in each timestep. The predicted 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, and *write* to M^v to update its information. 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 k_t. k_t is a representation of question q_t, and is then multiplied by a learned matrix M^k and softmaxed.

This creates a vector w_t, where entry j in w_t represents the correlation weight between the question q_t and memory slot j. This process of taking the dot product between an item embedding and a trainable matrix and softmaxing is similar to the concept of "attention", used in popular NLP techniques such as transformers [73].

Next, *read* from the value matrix by computing

$$r_t = \sum_{i=1}^{K} w_t(i) M^v(i). \tag{5.12}$$

Note that 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 master level of exercise q_t. Next, the item embedding k_t is appended to the read content r_t and fed forward through two linear layers. The first uses a tanh activation function, and the output p_t produced a single

node and a sigmoidal activation. In this way, the single value p_t represents the probability that the student will answer item q_t correctly at that timestep.

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 v_t. The manner in which M^v is updated is actually similar to that of an LSTM, allowing for "remembering" and "forgetting". Two trainable matrices are multiplied by v_t to produce an "erase" vector e_t and an "add" vector 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 1 get erased quite a bit. 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

$$M_t^v(i) = (M_{t-1}^v(i)[1 - w_t(i)e_t]) + w_t(i)a_t \tag{5.13}$$

Note that the correlation weights 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. Additionally, if a student answers each available item, then stacking each weights vector w_t into a matrix $W = \{w_t\}_{t=1}^L$ 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.

5.5.1 Deep-IRT

Deep-IRT, proposed by Chun-Kit Yeung [76] modifies the DKVMN architecture to allow a connection with Item Response Theory. Specifically, two separate feed forward layers are inserted, representing a student's k-th ability at time t θ_{tk} and concept difficulty β_k. Then the output probability is not another linear layer (as in DVKVMN), but is instead a function of θ_{tk} and β_k:

$$p_t = \frac{1}{1 + \exp\left(\beta_k - 3 \cdot \theta_{tk}\right)} \tag{5.14}$$

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 5.14, 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.

5.6 Self-Attentive Knowledge Tracing

In the field of natural language processing (NLP), the most state-of-the-art methods utilize a mechanism called self-attention [73], which rely on calculating the correlation between pairs of words in a sentence. Popular models such as BERT [27] and GPT-3 [11] 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 [52].

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 m_i. Additionally, like DKVMN, the current question q_t without the response is also embedded into a d-dimensional vector e_t.

The exercise embedding e_t is multiplied by a weights matrix to obtain a *query* vector $\boldsymbol{q}_t = W^Q e_t$. The interaction embedding m_i is used to create two vectors: a *key* vector $\boldsymbol{k}_i = W^K m_i$ and a *value* vector $\boldsymbol{v}_i = W^V m_i$.

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

$$a_t = \text{softmax}\left(\frac{K\boldsymbol{q}_t}{\sqrt{d}}\right)V \tag{5.15}$$

The value $\frac{K\boldsymbol{q}_t}{\sqrt{d}}$ yields a vector where each 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 a_t is sent through a few feed-forward layers, resulting in a vector $f_t = \text{FFN}(a_t)$. The output layer is $p_t = \sigma(f_t W + b)$, the probability that the student will answer the current exercise q_t correctly.

5.7 Performance Factors Analysis

An earlier approach to knowledge tracing was proposed by Pavlik et al. in 2009 with Performance Factors Analysis (PFA) [54]. 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 a student answering item i correctly is a linear combination of concept difficulty, previous successes, and previous failures:

$$p(j, k \in \mathrm{K}_i, s, f) = \sigma \left(\sum_{k \in K} (\beta_k + \gamma_k s_{jk} + \rho_k f_{jk} \right) \tag{5.16}$$

In Equation 5.16, 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

$$\begin{aligned} s_{jk} &= \sum_{i<t} \chi_{c_i=1} \cdot \chi_{k \in \mathrm{K}_i} \\ f_{jk} &= \sum_{i<t} \chi_{c_i=0} \cdot \chi_{k \in \mathrm{K}_i} \end{aligned} \tag{5.17}$$

where χ is the indicator function on some condition. 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 5.16 is essentially just a logistic regression. Note that similar to Deep-IRT, PFA focuses on the concept-level, rather than item-level, parameters.

5.7.1 Deep Performance Factors Analysis

Recent work has related PFA to the self-attention mechanism used in SAKT described in Section 5.6. Pu et al. [57] developed Deep Performance Factors Analysis (DPFA) and a new characterization of the weight parameters γ_k and ρ_k, using learned item embeddings e_i for each question q_i.

For the current question q_{t+1}, the attention between previous exercises is $A_{i,t+1} = e_i^\top e_{t+1}$, for $i \leq t$. 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:

$$w_i = \text{softmax}(A_{i,t+1} + d_{i,t+1}) \tag{5.18}$$

The mastery of knowledge concepts after a student completes interaction (q_i, c_i) is given as $v_i = [v_i^0, v_i^1] \in \mathbb{R}^2$. 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

$$p_{t+1} = \sigma\left(\beta_{t+1} + \sum_{i \leq t} \left(\chi_{c_i=0} \cdot w_i v_i^0 + \chi_{c_i=1} \cdot w_i v_i^1\right)\right) \tag{5.19}$$

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 5.16, 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 METHODS FOR KNOWLEDGE TRACING

The deep neural network approaches described in the previous section (including DKT, DKVMN, and SAKT) to the knowledge tracing problem have produced very high predictive power. Given a sequence previous student actions 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. But there already exists a theoretical framework for computing the probability of a correct response in Item Response Theory, introduced in Section 2.3.

In this section, we introduce an interpretable modification to deep knowledge tracing methods, reported in the publication [21]. 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. 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 Deep-IRT. 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 propsed 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 by producing estimates to discrimination and difficulty parameters. The focus here involves item-level parameters, rather than 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, similar to the methodology of ML2P-VAE in Section 3.1.

In this section, 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 AUC, IRT-inspired knowledge tracing provides an explicit representation of student knowledge Θ at each timestep. This representation of student knowledge is an upgrade from other deep knowledge tracing methods, which approximate skill mastery 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 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 $Q \in \{0,1\}^{n \times K}$ [24], 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.

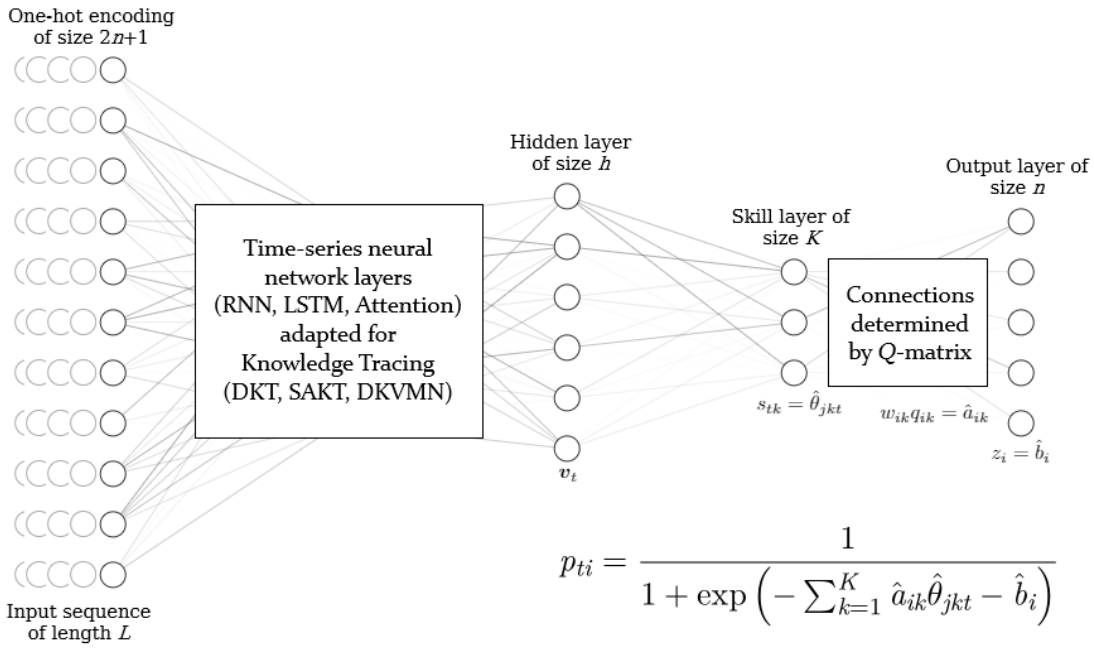

Figure 6.1: Visualization of integrating IRT into a knowledge tracing model with $L = 4$, $n = 5$, and $K = 3$.

Each student's response sequence includes $\boldsymbol{x}_0$, a null interaction to indicate the start of their interactions. A hyper-parameter 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 time-series neural network (TSNN), such as an LSTM (similar to DKT [55]) or an attention-based model (similar to SAKT [52]). This outputs a h-dimensional vector $\boldsymbol{v}_t$ for each interaction $\boldsymbol{x}_t$ in the input sequence.

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

Next, each $\boldsymbol{v}_t$ is sent through a linear layer feed-forward network with output size K (the number of latent concepts), yielding a 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 matrix W_s and 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 [34][23]. 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 estimates to difficulty parameters b_i. The student's k-th latent ability θ_k is estimated at timestep t via s_{tk}.

CHAPTER 7
COMPARISON OF KNOWLEDGE TRACING METHODS

7.1 Data Description

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.

Synth-5

[1] This dataset was generated by Piech et al. [55] 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 [44] 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, 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

[2] Sim200 differs from Synth-5 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 similar to the Sim-20 dataset described in Section 4.1, but the latent abilities Θ were generated according to a standard normal Gaussian distribution.

Statics2011

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

Assist2017

[4] The ASSISTments 2017 dataset contains real-world interactions from 1,709 students recorded on the ASSISTments online tutoring system. There is a large

[2]https://github.com/converseg/irt_data_repo/tree/master/sim200

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

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

Dataset	Items	Skills	Students	Interactions
Synth-5	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 datasets.

number of distinct items (4,117), and 102 latent concepts. Some items are tagged with the concept "noskill" – we treat this tag as a latent concept, otherwise all interactions involving such items would need to be thrown out.

7.2 Experiment Details

In the two simulated datasets (Synth-5 and Sim200), all students answering the same set of questions and thus all have the same length of response sequences (50 and 200, respectively). On Statics2011 and Assist2017, the maximum sequence length is set at $L = 128$, and student's whose response sequences are longer/shorter than 128 interactions have their response sequences wrapped/padded. The rest of the hyper-parameters are described in Table 7.2. The hyperparameters for DKT, SAKT, and DKVMN follow those reported in the corresponding literature.

7.3 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 Synth-5 and Sim200, the responses were generated with known IRT

Parameter	Synth-5	Sim200	Statics2011	Assist2017
max_len	50	200	128	128
input_size	101	201	1975	8235
output_size	50	200	987	4117
hid_size	64	64	50	100
skill_layer	5	20	61	102

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

Method	Synth-5	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: TestAUC values for various models on each dataset.

models which match the architecture of IRT-inspired methods.

Note that when working on Synth-5, 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 $W_p = Q$. 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 Synth-5 than seen in the Sim200 data.

When looking at the two real-world datasets, the trade-off in AUC is more significant, as it is not known if the student responses follow the ML2P model. There could also be inaccuracies in the given item-skill association Q-matrix, which our mod-

els are dependent on. Additional difficulties arise 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.

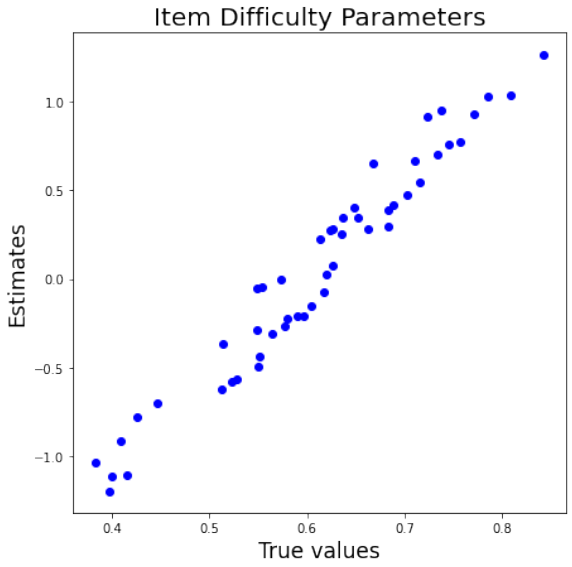

Figure 7.1: Correlation between DKT-IRT estimates and true values of Synth-5 item difficulty. SAKT-IRT produced similar results.

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. Due to the available public dataset, there is no access to the true values of student abilities Θ.

The parameter estimates can be directly compared to the true parameters in

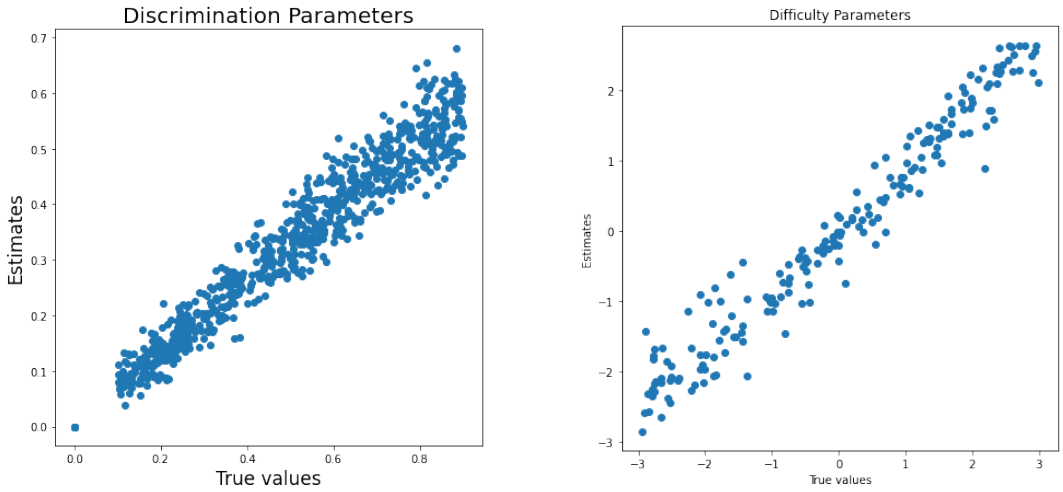

Figure 7.2: Correlation between true and estimated Sim200 (left) item discrimination parameters, and (right) item difficulty parameters.

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 Θ do not require any additional computations or transformations and are directly obtained from the nerual network. 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.

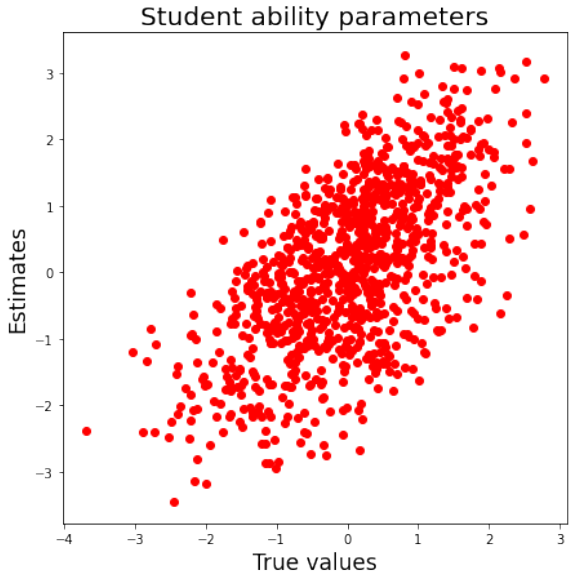

Figure 7.3: Correlation between true and estimated student ability parameters at $t = L = 200$ for the Sim200 dataset.

The explcit representation of knowledge Θ 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 Synth-5 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.

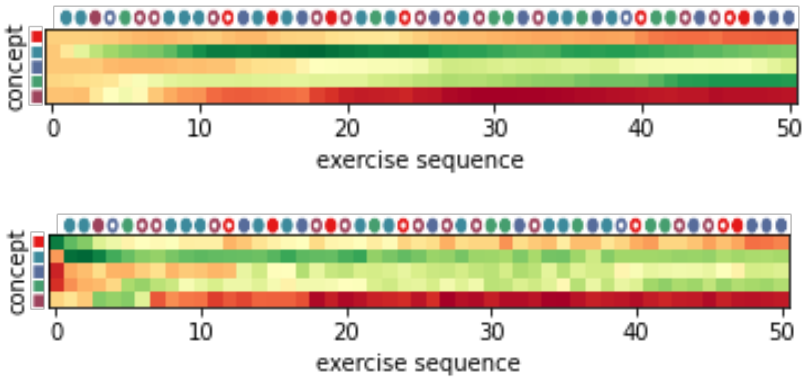

Figure 7.4: Tracing a student's knowledge mastery with DKT-IRT (top) and SAKT-IRT (bottom) as they progress through the items of the Synthetic-5 dataset.

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 recurrent structure and instead maintains connections to all previous interactions.

7.4 Discussion

The connection between IRT and knowledge tracing presented in this work introduces a trade-off between accuracy and interpretability. Further work to increase AUC to the level of DKVMN while maintaining explainability is worth exploring. 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 information 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.

Our 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. This 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.

CHAPTER 8
RELATED WORK

In this chapter, we introduce a few other application areas outside of education where the ML2P-VAE method can be applied. Although the development of the method was done with item response theory in mind, there are other fields which have similar goals to IRT where ML2P-VAE can be applied. As this is not the primary focus of this thesis, we introduce the application setting, draw analogues to education and IRT, and analyze some preliminary results.

8.1 Health Sciences

8.1.1 Beck Depression Inventory

-BDI is already connected to IRT -This is interesting because we can add in other features along with response data.

8.1.2 Personality Questionaires

Big 5, etc.

8.2 Sports Analytics

Consider cutting this section down a lot

Over the past few decades, sports franchises have become more willing to incorporate technology and analytics into their strategy, both on and off the field. For example, the large availability of data has influenced basketball teams to strive for more efficient offense Off the field, a common task is player evaluation. On sports talk shows, TV personalities often argue over who is the "greatest player of all

TODO: citation.

time". These arguments are driven by simple in-game measurements as well as more complicated analytics.

In 2011, the movie "Moneyball" brought the topic of sports analytics to the public eye, describing the strategy of the 2002 Oakland Athletics baseball team in finding the best valued players Oakland's approach was to find players who would reasonably increase the team's win total, but were undervalued in terms of salary. In general, the task of player evaluation seeks to quantify the contributions of players.

TODO: cite moneyball.

8.2.1 VAE for Player Evaluation

The ideas of the ML2P-VAE model can be used in this application. Though not all aspects of ML2P-VAE are relevant due to the lack of statistical theory (there is no analogue of IRT models in player evaluation), we can still interpret a hidden layer of the neural network. The goal of this application is to develop new measures for skill quantification of baseball players. This work was originally presented at the Conference on Fuzzy Systems and Data Mining (FSDM) 2019 by Converse et al. [19].

We use a modified VAE for the task of evaluating baseball players in four offensive skill areas. Baseball was chosen over other sports including football and basketball for various reasons: (a) the availability of data, (b) the popularity of analytics in the sport, (c) the consistency of rules and styles of play over the past 50 years, and (d) the lack of positional dependency.

The correspondence to IRT is as follows. Instead of responses to items on an exam, simple measurable statistics (hits, walks, etc.) over the course of a season. The

latent ability θ in IRT corresponds with the underlying skills required to produce the measurable statistics. In baseball, the four skills chosen are *contact* (how often does the batter hit the ball), *power* (how hard does the player hit the ball), *baserunning* (is the player good at running the bases), and *pitch intuition* (does the player swing when they are supposed to).

In ML2P-VAE, the core feature which allows for interpretation of the neural network is the Q-matrix, relating latent abilities with exam items. A similar binary matrix can be constructed associating underlying baseball skills with measurable statistics. For example, the statistic "home runs" requires only the power skill, while avoiding "strikeouts" requires both contact and pitch intuition. A full description of the game statistics used can be found in the appendix.

TODO: add Q-matrix and other stuff to appendix

This binary matrix determines the non-zero connections between the learned distribution layer (representing baseball skills) and the output layer (reconstructions of the input game statistics). Note that it is reasonable to use a VAE instead of regular autoencoder because it is assumed that among the population of professional baseball players, the distribution of each skill roughly follows a standard normal distribution. This assumption could be altered to allow for correlated underlying skills, but it would be difficult to find an accurate covariance matrix for abstract skills.

8.2.2 Experiments

Should add citation to data

Data was gathered from Major League Baseball players from 1950-2018, yielding 8,604 samples, where each data point corresponds to a particular player's perfor-

mance in a particular year. 13 measurable game statistics were chosen as inputs to the VAE: singles (1B), doubles (2B), home runs (HR), runs (R), runs batted in (RBI), walks (BB), intentional walks (IBB), strikeouts (K), sacrifice (SAC), grounded into double play (GDP), stolen bases (SB), caught stealing (CS), and walk/strikeout ratio (BB/K). Each statistic was rescaled using Gaussian normalization so that each input feature was centered at 0 with variance one. Additionally, the game statistics strikeouts and caught stealing were multiplied by -1 for each observation so that a larger number is more desirable. As in the ML2P-VAE architecture, the decoder weights are constrained to be non-negative, so that a higher skill value can only increase the reconstructed in-game statistics.

Since this is an unsupervised method with the goal of creating *new* skill measures, it is difficult to evaluate the obtained results. Instead, we take the more commonly used baseball statistics contact rate (CR), speed score (SPD), isolated power (ISO), and on-base percentage (OBP) to compare with the skills *contact*, *baserunning*, *power*, and *pitch intuition*, respectively. More informationon how these measures are calculated can be found in the appendix.

TODO: add to appendix

Each skill is plotted against its evaluation statistic in Figure 8.1 [19]. Note that each of the four plots display high correlation, but do not match exactly. This is desirable in the sense that our new skill quantifications do in fact measure what they are intended to measure, but give new insights and ranking for each player. Though these results could be improved, it is clear that variations of the ML2P-VAE model can be applied in areas other than education.

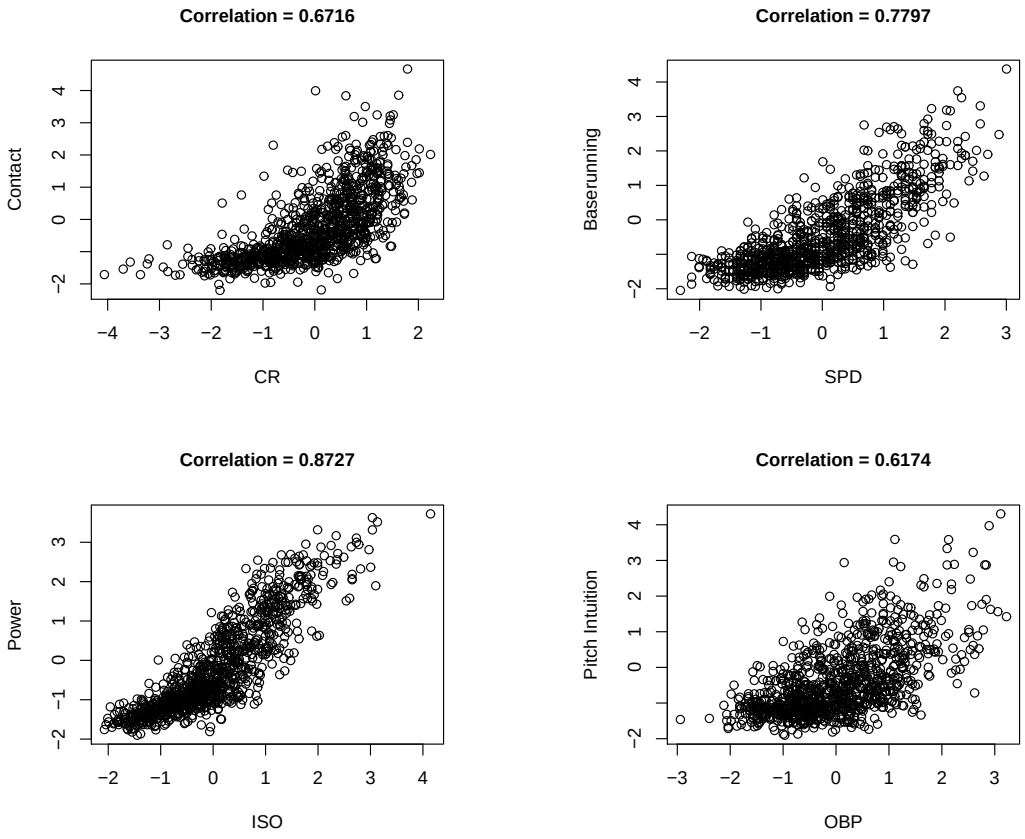

Figure 8.1: Each latent skill's estimates plotted against its evaluation statistic.

APPENDIX A
GLOSSARY OF NOTATION

abc

APPENDIX B
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 [8]. 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.

B.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 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 B.1 which takes inputs with 10 features and classifies into three categories [43]. 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 [49].

Edit this image with correct notation

While the architecture of a neural network can be described using the lens of graph theory, the inner workings are better described 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

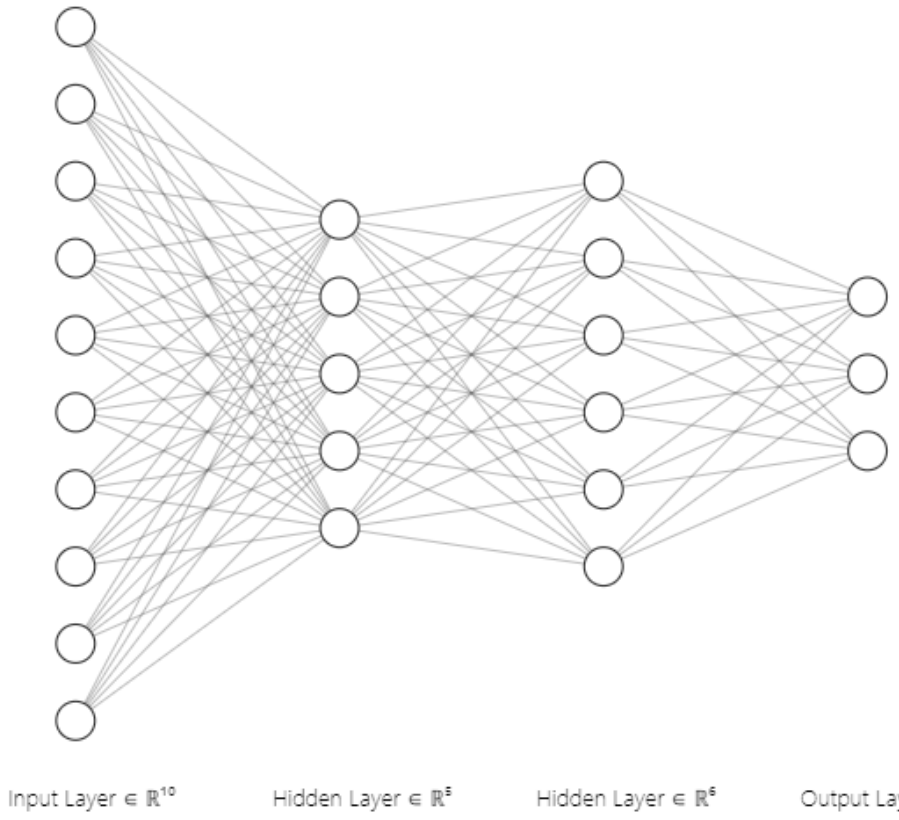

Figure B.1: A basic FFN with input size $n_0 = 10$ and output size $n_3 = 3$, with two hidden layers of size $n_1 = 5$ and $n_2 = 6$.

transformation followed by a nonlinear re-scaling. For a given input vector $\boldsymbol{x}^* \in \mathbb{R}^{n_0}$ and its corresponding true label $\boldsymbol{y}^*$, the value in the first hidden layer is calculated as

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

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 B.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 $\boldsymbol{z}_1^* = W_1\boldsymbol{x}^* + \boldsymbol{b}_1 \in \mathbb{R}^{n_1}$ elementwise. Examples of different activation functions are given in Appendix B.2. Matrix notation can also be abandoned by writing the activation of the i-th node in layer $l = 1$ as

$$a_{1,i}^* = f_1\left(b_{1,i} + \sum_{j=1}^{n_0} w_{ij}^1 \cdot x_j^*\right) \tag{B.2}$$

where w_{ij}^1 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

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

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 is often a one-hot encoding, so the prediction $\hat{\boldsymbol{y}}^*$ should be a probability distribution each entry describes the certainty of 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% sure that the input features are that of a cat, 32% sure that the input is a dog, and 3% sure that the input is a bird.

B.2 Activation Functions

The main purpose of activation functions is to rescale each layer so that every activation value falls in the same range. Doing multiple matrix multiplications in a row can easily cause values to become very large, and leading to overfitting and other complications [68]. 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 applied to a vector elementwise independently, except for the softmax function which maps an n-dimensional vector to an n-dimensional probability distribution.

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

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

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

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

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

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 $\frac{d\sigma}{dz}\Big|_{z=z_0}$ 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.

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

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 B.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 can be understood as a discrete probability distribution. This can be particularly useful in the output layer for a multi-class classification application.

B.3 Optimization and Backpropagation

Figure **??** and Equations B.1 and B.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 "train" 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 [31], consider the simple mean squared-error loss function

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

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

Recall that if an input (or set of inputs) $\boldsymbol{x}^*$ is fixed, then the prediction 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 B.8.

Though deep neural networks can have thousands, millions, or billions of parameters [11], calculating gradients remains feasible because of the backpropagation algorithm [63]. While obtaining predictions from a neural networks works in a left-to-right fashion ($\boldsymbol{a}_3$ depends on $\boldsymbol{a}_2$ depends on $\boldsymbol{a}_1$ 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 B.3, we can write

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

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^3_{ij} used in Equation B.9. We have

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

Notice how in Equation B.10, information first calculated in Equation B.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 $\boldsymbol{x}$:

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

where $\boldsymbol{\Lambda}$ is a vector containing all trainable parameters W_l and $\boldsymbol{b}_l$ and η is the learning rate hyperparameter [65]. This process is repeated with different inputs from the training set $\boldsymbol{x}$.

TODO: could give more on SGD, adam, etc

APPENDIX C
ML2PVAE CODE EXAMPLES

APPENDIX D
DETAILS ON RELATED WORKS

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 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] 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.

[6] 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.

[7] 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.

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

[9] 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.

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

[11] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 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.

[12] Gary Buck and Kikumi Tatsuoka. Application of the rule-space procedure to language testing: examining attributes of a free response listening test. *Language Testing*, 15(2):119–157, 1998.

[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] G. Converse, M. Curi, and S. Oliveira. Autoencoders for educational assessment. In *International Conference on Artificial Intelligence in Education (AIED)*, 2019.

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

[19] 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.

[20] 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.

[21] Geoffrey Converse, Suely Oliveira, and Shi Pu. Incorporating item response theory into knowledge tracing. In *International Conference on Artificial Intelligence in Education (AIED)*, 2021.

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

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

[24] 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.

[25] 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.

[26] 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.

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

[28] Carl Doersch. Tutorial on variational autoencoders, 2016.

[29] 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.

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

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

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

[33] 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.

[34] 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.

[35] 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.

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

[37] 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.

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

[39] 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.

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

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

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

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

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

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

[46] 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.

[47] 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.

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

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

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

[51] Christopher Olah. Understanding lstm networks, 2015.

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

[53] 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.

[54] 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.

[55] 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.

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

[57] Shi Pu, Geoffrey Converse, and Yuchi Huang. Deep performance factors analysis for knowledge tracing. In *International Conference on Artificial Intelligence in Education (AIED)*, 2021.

[58] 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.

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

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

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

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

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

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

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

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

[67] 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.

[68] 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.

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

[70] 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.

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

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

[73] 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.

[74] 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.

[75] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stefan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, lhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antonio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. Scipy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 2020.

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

[77] 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.

[78] 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.

[79] 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.