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

Date?

ACKNOWLEDGEMENTS

ABSTRACT

PUBLIC ABSTRACT

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

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 4-6 of Part II explore a task common in electroninc learning environments: 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.

CHAPTER 1
ITEM RESPONSE THEORY BACKGROUND

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 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 true score theory [30]. But there are clear issues with this approach. Not all questions on an exam or homework assignment is 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. 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 studied which attempt to more accurately quantify student learning. Cognitive Diagnostic Models (CDM) (TODO: citation) aim to classify whether students possess mastery of a given skill or not. This discrete classification can be useful in determining whether or not a student meets a prerequisite, or deciding whether or not they are ready to move on to the next level of coursework. We focus instead on Item Response Theory, where student knowledge is assumed to be continuous.

1.1 Item Response Theory

Item Response Theory (IRT) is a field of quantitative psychology which uses statistical models to model student ability [23]. 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"or "skill." Often, it is assumed that amongst the population of students, $\theta_j \sim \mathcal{N}(0,1)$ [30].

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 $\Theta_j = (\theta_{1j}, \theta_{2j}, \ldots, \theta_{Kj})^\top$. 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.

Note that Θ_j is not directly observable in any way. Instead, a common goal is to infer student's knowledge Θ_j from on their responses on some assessment containing n questions, referred to as items. A student's set of responses can be written as a binary n-dimensional vector $\vec{u}_j = (u_{1j}, u_{2j}, \ldots, u_{nj})^\top$, where

$$u_{ij} = \begin{cases} 1 & \text{if student } j \text{ answers item } i \text{ correctly} \\ \\ 0 & \text{otherwise} \end{cases} \tag{1.1}$$

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 Θ_j:

$$P(u_{ij} = 1|\Theta_j) = f(\Theta_j; V_i) \tag{1.2}$$

where V_i is a set of parameters associated with item i. In general, $f : \mathbb{R}^K \to [0, 1]$ is some continuous function which is strictly increasing with respect to Θ_j.

In the following sections, we describe various candidates for the function f. Though each is presented in the context of single-dimensional IRT ($K = 1$), they can all be easily adapted to higher dimensions.

1.1.1 Rasch Model

One of the first models was proposed by Georg Rasch in 1960. 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 $\frac{\xi}{\xi+\delta} = \frac{1}{1+\delta/\xi}$ 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

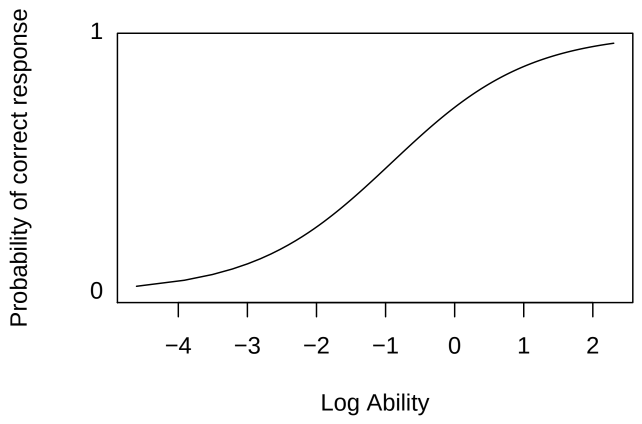

Figure 1.1: An item characteristic curve visualizes the relation between a student's ability and the probability of answering an item correctly.

Model, often referred to as the Rasch Model.

$$P(u_{ij} = 1|\theta_j; b_i) = \frac{1}{1 + e^{b_i - \theta_j}} \tag{1.3}$$

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. A plot of Equation 1.3 for a fixed item (fixed b_i) is shown in Figure 1.1. The horizontal axis represents $\log \theta$, and the vertical axis represents $P(u_{ij} = 1|\theta, b_i)$. This type of graph is often referred to as an item characteristic curve (ICC).

1.1.2 Normal Ogive Model

*This section isn't very important to rest of work

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 / have not mastered the knowledge concept θ [30]. In other words, a_i tells *how much* of skill θ is required to answer item i correctly.

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

$$P(u_{ij} = 1|\theta_j; a_i, b_i) = \frac{1}{\sqrt{2\pi}} \int_{-a_i\theta_j+b_i}^{\infty} e^{\frac{-z^2}{2}} dz \tag{1.4}$$

Note the similarity between Equation 1.4 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.

1.1.3 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

$$P(u_{ij} = 1|\theta_j; a_i, b_i) = \frac{1}{1 + e^{-a_i\theta_j+b_i}} \tag{1.5}$$

Equation 1.5 has the same form as that of the Rasch model in Equation 1.3, but adds in the discrimination parameter a_i. 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 [2].

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 1.5, using this model makes parameter estimation much easier.

1.1.4 Multidimensional Item Response Theory

The previously described statistical models can all be extended so that each student possesses K latent traits. In multidimensional item response theory (MIRT), models give the probability of a correct answer as a function of the student ability vector $\Theta = (\theta_1, \ldots, \theta_K)^\top$. The generalization of 1.5 is given by the multidimensional logistic 2-parameter (ML2P) model:

$$P(u_{ij} = 1|\Theta_j; \vec{a_i}, b_i) = \frac{1}{1 + \exp\left(-\vec{a_i}^\top \Theta_j + b_i\right)} = \frac{1}{\exp\left(-\sum_{k=1}^{K} a_{ik}\theta_{kj} + b_i\right)} \tag{1.6}$$

Here, the discrimination parameters $\vec{a_i} \in \mathbb{R}^K$ are given as vector, where each entry $a_{ik} \in \vec{a_i}$ quantifies *how much* of skill k is required to answer item i correctly. The ML2P model is the main focus of this thesis.

TODO: mention MDISC and how this scales

In MIRT, it is convenient to notate the relationship between skills and items with binary matrix. Define the Q-matrix [15] $Q \in \{0, 1\}^{n \times K}$ so that

$$q_{ik} = \begin{cases} 1 & \text{if item } i \text{ requires skill } k \\ \\ 0 & \text{otherwise} \end{cases}. \tag{1.7}$$

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 1.6), notice that if $q_{ik} = 0$, then $a_{ik} = 0$ as well. Though experts can produce a Q-matrix for a given assessment, the matrix of discrimination parameters $(a_{ik})_{i,k}$ can not be discovered so easily.

*How in-depth do these descriptions need to be? Could just give overall idea and describe weaknesses

1.2 IRT Parameter Estimation

1.2.1 Maximum Likelihood Estimation

TODO: item parameter estimation

TODO: ability parameter estimation

1.2.2 Joint Maximum Likelihood Estimation

1.2.3 Marginal Maximum Likelihood Estimation

TODO: MMLE

TODO: EM

1.3 Artificial Neural Networks

In recent years, artifical neural networks (ANN) have become an increasingly popular tool for machine learning problems. Though they have been around since the 1960's (TODO: citation), GPU technology has become more accessible and modern computers are more powerful, allowing anyone interested to train a basic neural network on their 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 (TODO: citations).

One of the biggest critiques of ANN is their black-box nature, meaning that the decision process that a trained model uses 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 predicts a particular data sample the way that it does. For example, if a financial institution is using data science 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. Most customers will not be satisfied with "the computer told us so," and there is a possibility that a black-box neural network could learn and use features such as race or gender in its prediction, which is illegal in the United States (TODO: definitely need citation or delete).

The push for explainable AI have led researchers down two paths. One group has tried to incorporate deep learning methods with existing interpretable methods, in hopes of increasing the performance of explainable methods without sacrificing its interpretability (TODO: citation). 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 [27]. 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 understand the features extracted in the hidden layers [36].

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 failed a test, and a teacher should be given instructions on *how* to fix the student's misconceptions.

1.3.1 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 $x \in \mathbb{R}^n$, define the *encoder* as a function $f : \mathbb{R}^n \to \mathbb{R}^m$ mapping $x \mapsto z := f(x)$. Usually, $m < n$, and 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 $g : \mathbb{R}^m \to \mathbb{R}^n$ mapping $z \mapsto \hat{x} := g(z)$. The decoder maps a hidden representation 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 $\mathcal{A}(x) := g(f(x)) : \mathbb{R}^n \to \mathbb{R}^n$. 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

$$\mathcal{L}(x) = ||x - g(f(x))||_2^2 \tag{1.8}$$

or cross-entropy loss for binary data

$$\mathcal{L}(x) = \sum_{i=1}^{n} -x_i \log(g(f(x_i))) - (1 - x_i)\log(1 - g(f(x_i))). \tag{1.9}$$

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 [26]. AEs have clear applications in image compression straight out-of-the-box, and can be modified for more complicated problems. Denoising autoencoders [32] are capable of processing noisy images and cleaning them up. To do this, they corrupt input data by deleting pixels at random and reconstructing the original image. Autoencoders can also be modified for data generation applications using a variational autoencoder.

*relevant sources: [17] [21] [4], infoVAE, ELBO, "towards deeper understanding of VAE"

1.3.2 Variational Autoencoders

The motivation for designing a variational autoencoder (VAE), introduced by Kingma and Welling [21], 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 $X = \{\boldsymbol{x}_i\}_{i=1}^N$, 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 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 $p(\boldsymbol{x}) = \int p(\boldsymbol{z})\boldsymbol{x}|\boldsymbol{z})d\boldsymbol{z}$ 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.

1.3.2.1 A note on Information Theory

Before continuing, we introduce some ideas from information theory: entropy, cross-entropy, and KL-Divergence [3]. 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

$$H[p] = -\int p(x) \log p(x) dx \tag{1.10}$$

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

$$H[p, q] = -\int p(x) \log q(x) dx \tag{1.11}$$

We can simply define *Kullback-Leibler Divergence* (KL-Divergence) [22] as

$$\mathcal{D}_{KL}\left[p||q\right] = H[p] - H[p, q] = -\int p(x) \log\left(\frac{q(x)}{p(x)}\right) dx \tag{1.12}$$

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 $p(x) = q(x)$.

1.3.2.2 VAE Derivation

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

$$\log p_\beta(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) = \sum_{i=1}^{N} \log p_\beta(\boldsymbol{x}_i) \tag{1.13}$$

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

$$\begin{aligned}
\log p_\beta(\boldsymbol{x}) &= \int q_\alpha(\boldsymbol{z}|\boldsymbol{x}) \log p_\beta(\boldsymbol{x}) dz \\
&= \int q_\alpha(\boldsymbol{z}|\boldsymbol{x}) \log \left(\frac{p_\beta(\boldsymbol{z}|\boldsymbol{x}) p_\beta(\boldsymbol{x})}{p_\beta(\boldsymbol{z}|\boldsymbol{x})} dz \right) \\
&= \int q_\alpha(\boldsymbol{z}|\boldsymbol{x}) \log \left(\frac{p_\beta(\boldsymbol{x}, \boldsymbol{z})}{p_\beta(\boldsymbol{z}|\boldsymbol{x})} \right) dz \\
&= \int q_\alpha(\boldsymbol{z}|\boldsymbol{x}) \left(\log \frac{q_\alpha(\boldsymbol{z}|\boldsymbol{x})}{p_\beta(\boldsymbol{z}|\boldsymbol{x})} + \log \frac{p_\beta(\boldsymbol{x}, \boldsymbol{z})}{q_\alpha(\boldsymbol{z}|\boldsymbol{x})} \right) dz \qquad (1.14) \\
&= \mathcal{D}_{KL} \left[q_\alpha(\boldsymbol{z}|\boldsymbol{x}) || p_\beta(\boldsymbol{z}|\boldsymbol{x}) \right] + \int q_\alpha(\boldsymbol{z}|\boldsymbol{x}) \log \left(\frac{p_\beta(\boldsymbol{x}, \boldsymbol{z})}{q_\alpha(\boldsymbol{z}|\boldsymbol{x})} \right) dz \\
&= \mathcal{D}_{KL} \left[q_\alpha(\boldsymbol{z}|\boldsymbol{x}) || p_\beta(\boldsymbol{z}|\boldsymbol{x}) \right] + \mathbb{E}_{q_\alpha(\boldsymbol{z}|\boldsymbol{x})} \left[-\log q_\alpha(\boldsymbol{z}|\boldsymbol{x}) + \log p_\beta(\boldsymbol{x}, \boldsymbol{z}) \right] \\
&= \mathcal{D}_{KL} \left[q_\alpha(\boldsymbol{z}|\boldsymbol{x}) || p_\beta(\boldsymbol{z}|\boldsymbol{x}) \right] + \tilde{\mathcal{L}}(\alpha, \beta; \boldsymbol{x})
\end{aligned}$$

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

$$\log p_\beta(\boldsymbol{x}) \geq \tilde{\mathcal{L}}(\alpha, \beta; \boldsymbol{x}) = -\mathcal{D}_{KL} \left[q_\alpha(\boldsymbol{z}|\boldsymbol{x}) || p_\beta(\boldsymbol{z}) \right] + \mathbb{E}_{q_\alpha(\boldsymbol{z}|\boldsymbol{x})} \left[\log p_\beta(\boldsymbol{x}|\boldsymbol{z}) \right] \qquad (1.15)$$

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 1.15, 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 2.1.1, 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 [17] that the KL-Divergence between an independent Gaussian distribution and a standard normal distribution of dimension K is calculated as

Should I show this?

$$\mathcal{D}_{KL}\left[\mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\sigma}_0^2 I)||\mathcal{N}(0, I)\right] = \frac{1}{2}\sum_{k=1}^{K}\left(\mu_k^2 + \sigma_{0,k}^2 - 1 - \log(\sigma_{0,k}^2)\right) \tag{1.16}$$

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 1.16 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 1.15 is similar to Equation 1.11, 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 $L = 1$ [21]. So then

$$\begin{aligned}\mathbb{E}_{q_\alpha(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\beta(\boldsymbol{x}|\boldsymbol{z})\right] &\approx \frac{1}{L}\sum_{l=1}^{L}\log p_\beta(\boldsymbol{x}|\boldsymbol{z}^{(l)}) \\ &\approx \log p_\beta(\boldsymbol{x}|\boldsymbol{z}^*) \\ &= \log\left(\prod_{i=1}^{n} p_\beta(x_i = 1|\boldsymbol{z}^*)^{x_i}\cdot p_\beta(x_i = 0|\boldsymbol{z}^*)^{x_i}\right) \\ &= \sum_{i=1}^{n} x_i \log \hat{x}_i + (1-x_i)\log(1-\hat{x}_i)\end{aligned} \tag{1.17}$$

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 1.17 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 1.16 and Equation 1.17, we can easily compute the objective function $\tilde{\boldsymbol{\mathcal{L}}}(\alpha, \beta; \boldsymbol{x}_0)$.

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

$$\begin{aligned}\boldsymbol{\mathcal{L}}(\alpha, \beta; \boldsymbol{x}) &= -\tilde{\boldsymbol{\mathcal{L}}}(\alpha, \beta; \boldsymbol{x}) \\ &= \Big(\sum_{i=1}^{n} -x_i \log p_\beta(x_i|\boldsymbol{z}^*) - (1-x_i)\log(1-p_\beta(x_i|\boldsymbol{z}^*))\Big) + \mathcal{D}_{KL}\left[q_\alpha(\boldsymbol{z}|\boldsymbol{x})||p_\beta(\boldsymbol{z})\right]\end{aligned} \tag{1.18}$$

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 2.6. Note that $\boldsymbol{\mathcal{L}}(\alpha, \beta; \boldsymbol{x})$ 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.

1.3.2.3 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 $p_\beta(\boldsymbol{z}) = \mathcal{N}(0, I)$. The first d nodes represent the latent mean vector $\boldsymbol{\mu}$, and the last d nodes represent the latent variances $\boldsymbol{\sigma}^2$. 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 $p_\beta(\boldsymbol{x}|\boldsymbol{z})$ hase 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 $\hat{\boldsymbol{x}_0}$.

The VAE architecture is summarized in Figure 1.2. Note that the VAE does not need to be symmetric; the encoder and decoder can have a different number of hidden layers of different sizes.

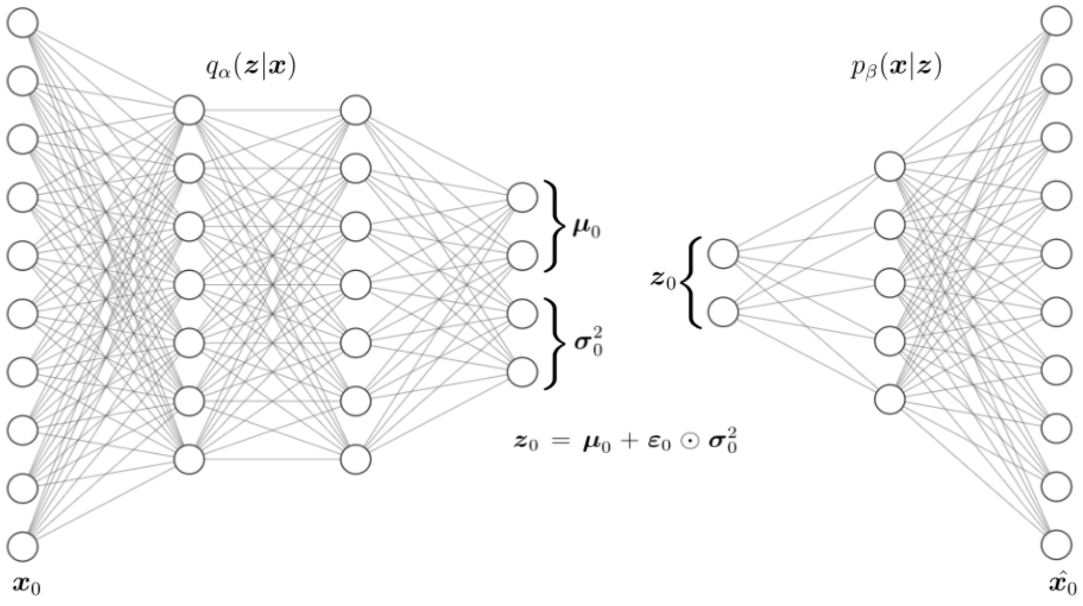
$q_\alpha(\boldsymbol{z}|\boldsymbol{x})$
$p_\beta(\boldsymbol{x}|\boldsymbol{z})$
$\boldsymbol{\mu}_0$
$\boldsymbol{\sigma}_0^2$
$\boldsymbol{z}_0$
$\boldsymbol{z}_0 = \boldsymbol{\mu}_0 + \boldsymbol{\varepsilon}_0 \odot \boldsymbol{\sigma}_0^2$
$\boldsymbol{x}_0$
$\hat{\boldsymbol{x}}_0$

Figure 1.2: Visualization of a VAE architecture with $n = 10$ and $d = 2$.

CHAPTER 2
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.

2.1 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 1.1. 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 1.7.

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 $\vec{u}_j$ 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 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 hase n nodes, and $Q \in \{0,1\}^{n \times K}$. Further, require the use of the sigmoidal activation function

$$\sigma(z_i) = \frac{1}{1+e^{z_i}} \tag{2.1}$$

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 2.1 and Equation 1.6. The constraint on the weights along with the sigmoidal 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 $\vec{u}_j$) can be interpreted as estimates to the student ability parameter θ_{kj}.

may want to mention that the value outputed by encoder is then sampled from for dceoder 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 $-\theta a_{ik} = \theta(-a_{ik})$

2.1.1 Full Covariance Matrix Implementation

There are many publicly available code exmples 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-Liebler 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 correpond to a probability distribution. For a K-dimensional multivariate Gaussian distribution, these values are a vectore $\mu_0 \in \mathbb{R}^K$ 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 [1]. The sample of the multivariate Gaussian is calculated as $z_0 = \mu_0 + G_0 \vec{\varepsilon_0}$, where $\vec{\varepsilon_0} = (\varepsilon_1, \ldots, \varepsilon_K)^\top$ and $\varepsilon_i \sim \mathcal{N}(0, 1)$.

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

$$\begin{aligned}&\mathcal{D}_{KL}\left[\mathcal{N}(\mu_0, \Sigma_0)||\mathcal{N}(\mu_1, \Sigma_1)\right] = \\ &\frac{1}{2}\left(\operatorname{tr}(\Sigma_1^{-1})\Sigma_0) + (\mu_1 - \mu_0)\Sigma_1^{-1}(\mu_1 - \mu_0) - K + \ln\left(\frac{\det \Sigma_1}{\det}\Sigma_0\right)\right)\end{aligned} \tag{2.2}$$

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 2.2 computes $\ln \det \Sigma_0$, so we must ahve $\det \Sigma_0 > 0$ 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 2.1. 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 nonsinguar, 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 symmteric 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 2.1 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.

2.1.2 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 $\hat{\Sigma} \in \mathbb{R}^{K\times K}$:

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

$$\hat{\Sigma}_{kl} = \frac{\sum_{i=1}^{N}(k_i - \bar{k})(l_i - \bar{l})}{\sqrt{\sum_{i=1}^{N}(k_i - \bar{k})^2}\sqrt{\sum_{i=1}^{N}(l_i - \bar{l})^2} \tag{2.3}$$

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 1.3, rather than the ML2P model as in Equation 1.6, 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 1.3.

2.1.3 On Convergence

Can't really guarantee convergence globally with SGD because neural nets are super nonconvex

2.1.3.1 Proving local minimum at true solution

We define the true and predicted probability of student j answering item i correctly with P_{ij} and $\hat{P}_{ij}$, respectively. The former comes from the ML2P model, and the latter is the output of a neural network. We assume the more simple case, where $\Theta \sim \mathcal{N}(0, I)$, rather than $\mathcal{N}(\mu, \Sigma)$.

$$P_{ij} = \frac{1}{1 + \exp\left(-\sum_{k=1}^{K} a_{ik}\theta_{jk} + b_i\right)} \tag{2.4}$$

$$\hat{P}_{ij} = \frac{1}{1 + \exp\left(-\sum_{k=1}^{K} \hat{a}_{ik}(\hat{\theta}_{jk} + \varepsilon_k \hat{\sigma}_k) + \hat{b}_i\right)} \tag{2.5}$$

Note that P_{ij} is unknown, and we instead have a response sequence $\vec{u}_j = (u_{1j}, \ldots, u_{nj})^\top$ with $u_{ij} = \text{Bern}(P_{ij})$. This also means that $\mathbb{E}[u_{ij}] = P_{ij}$. The variables $\hat{a}_{ik}$, $\hat{b}_i$, $\hat{\theta}_{ik}$, and are parameter estimates from the VAE. The first two are parameters in the neural network, and the ability estimates are taken from feeding responses to the encoder, i.e., $\hat{\Theta}_j = \text{Encoder}(\vec{u}_j)$. The noise $\varepsilon = (\varepsilon_k)_{1\le k\le K} \sim \mathcal{N}(0, I)$ is introduced by the sampling operation in the VAE.

Instead of VAE loss (which is really ELBO), this should be the marginal prob. of data (see Kingma/Welling)

The loss function for a VAE is given by

$$\begin{aligned}\mathcal{L}(\vec{u}_j) &= -\sum_{i=1}^{n}\Big[u_{ij}\log(\hat{P}_{ij}) + (1-u_{ij})\log(1-\hat{P}_{ij})\Big] + \mathbb{E}_{q_\alpha(\hat{\theta}|\vec{u}_j)}\log\left(\frac{q_\alpha(\hat{\theta}|\vec{u}_j)}{p(\theta)}\right) \\ &= \mathcal{L}_{\text{REC}} + \mathcal{L}_{\text{KL}}\end{aligned} \tag{2.6}$$

We break up the VAE loss into two terms, the reconstruction loss $\mathcal{L}_{\text{REC}}$ and the KL-divergence loss $\mathcal{L}_{\text{KL}}$. in the latter, the distribution $q_\alpha(\hat{\theta}|\vec{u}_j)$ is the output of the encoder, and $p(\theta)$ is the assumed prior distribution of Θ, which we set to be $\mathcal{N}(0, I)$.

We write $P_{ij} = \mathbb{E}(u_{ij})$, and similarly define a new "expected" loss function:

$$\begin{aligned}\mathcal{L}_{\mathbb{E}}(P_j) &= \mathbb{E}_{u_j}[\mathcal{L}(u_{:j})] \\ &= -\sum_{i=1}^{n}[P_{ij}\log(\hat{P}_{ij}) + (1-P_{ij})\log(1-\hat{P}_{ij})] + \mathbb{E}_{q_\alpha(\hat{\theta}|u_j)}\log\left(\frac{q_\alpha(\hat{\theta}|u_j)}{p(\theta)}\right) \\ &= \mathcal{L}_{\mathbb{E}[\text{REC}]} + \mathcal{L}_{\mathbb{E}[\text{KL}]}\end{aligned} \tag{2.7}$$

Notice that calculation of this "expected loss" requires the unknown $P_{:j}$. But when we have large amounts of data, we can think of P_{ij} as the average value of the response u_{ij}, so using this unknown value here is justified.

Define $z_i = a_{i:} \cdot \theta_{j:} - b_i$ and $\hat{z}_i = \hat{a}_{i:} \cdot (\hat{\theta}_{j:} + \varepsilon \cdot \hat{\sigma}) - \hat{b}_i$. Note that z_i is fixed, dependent on the data, and does not depend on any parameters of the neural network. $\hat{z}_:$ is the input to the final layer of the decoder, and the VAE output is

$\hat{P}_{ij} = \sigma(\hat{z}_i)$, where $\sigma(\cdot)$ is the sigmoidal activation function. We compute derivatives of the expected loss function, looking individually at the reconstruction and KL terms.

$$\begin{aligned}\frac{\partial \mathcal{L}_{\mathbb{E}[\text{REC}]}}{\partial \hat{z}_i} &= \frac{-1}{1+e^{-z_i}} \cdot \frac{1}{1+e^{\hat{z}_i}} - \frac{1}{1+e^{z_i}} \cdot \frac{-1}{1+e^{-\hat{z}_i}} \\ &= \frac{-1}{(1+e^{-z_i})(1+e^{\hat{z}_i})} + \frac{1}{(1+e^{z_i})(1+e^{-\hat{z}_i})} \\ &= \frac{-1}{(1+e^{-a_{i:}\cdot\theta_{j:}+b_i})(1+e^{\hat{a}_{i:}\cdot(\hat{\theta}_{j:}+\varepsilon_{:}\hat{\sigma}_{:})-\hat{b}_i})} \\ &\quad + \frac{1}{(1+e^{a_{i:}\cdot\theta_{j:}-b_i})(1+e^{-\hat{a}_{i:}\cdot(\hat{\theta}_{j:}+\varepsilon_{:}\hat{\sigma}_{:})+\hat{b}_i})}\end{aligned} \tag{2.8}$$

$$\begin{aligned}\frac{\partial \mathcal{L}_{\mathbb{E}[\text{REC}]}}{\partial \hat{a}_{ik}} &= \frac{\partial \mathcal{L}_{\mathbb{E}[\text{REC}]}}{\partial \hat{z}_i}\frac{\partial \hat{z}_i}{\partial \hat{a}_{ik}} = \frac{\partial \mathcal{L}_{\mathbb{E}[\text{REC}]}}{\partial \hat{z}_i}(\hat{\theta}_{jk}+\varepsilon_k\hat{\sigma}_k) \\ \frac{\partial \mathcal{L}_{\mathbb{E}[\text{REC}]}}{\partial \hat{b}_i} &= \frac{\partial \mathcal{L}_{\mathbb{E}[\text{REC}]}}{\partial \hat{z}_i}\frac{\partial \hat{z}_i}{\partial \hat{b}_i} = \frac{\partial \mathcal{L}_{\mathbb{E}[\text{REC}]}}{\partial \hat{z}_i}(-1) \\ \frac{\partial \mathcal{L}_{\mathbb{E}[\text{REC}]}}{\partial \hat{\theta}_{ik}} &= \frac{\partial \mathcal{L}_{\mathbb{E}[\text{REC}]}}{\partial \hat{z}_i}\frac{\partial \hat{z}_i}{\partial \hat{\theta}_{ik}} = \frac{\partial \mathcal{L}_{\mathbb{E}[\text{REC}]}}{\partial \hat{z}_i}(\hat{a}_{ik})\end{aligned} \tag{2.9}$$

Rather than setting these to zero and solving, we show that the most intuitive solution, $\hat{a}_{ik} = a_{ik}$, $\hat{b}_i = b_i$, and $\hat{\theta}_{jk} = \theta_{jk}$, is in fact a minimum of the expected loss function. But first, we must take another expectation over the random variable $\varepsilon \sim \mathcal{N}(0, I)$. Obviously, we have that $\mathbb{E}[\varepsilon_k] = 0$; this makes our calculations very simple. Notice that we have

$$\begin{aligned}&\mathbb{E}_{\varepsilon}\left[\frac{\partial \mathcal{L}_{\mathbb{E}[\text{REC}]}}{\partial \hat{z}_i}\right]\Big|_{\hat{a}_{ik}=a_{ik},\hat{b}_i=b_i,\hat{\theta}_{jk}=\theta_{jk}} \\ &= \frac{-1}{(1+e^{-a_{ik}\theta_{jk}+b_i})(1+e^{a_{ik}(\theta_{kj}+0\cdot\hat{\sigma}_k)-b_i})} + \frac{1}{(1+e^{a_{ik}\theta_{jk}-b_i})(1+e^{-a_{ik}(\theta_{jk}+0\cdot\hat{\sigma}_k)+b_i})} \\ &= 0\end{aligned} \tag{2.10}$$

Therefore we clearly have

$$\begin{aligned}
&\mathbb{E}_\varepsilon\left[\frac{\partial \mathcal{L}_{\mathbb{E}[\text{REC}]}}{\partial \hat{a}_{ik}}\right]\Big|_{\hat{a}_{ik}=a_{ik},\hat{b}_i=b_i,\hat{\theta}_{jk}=\theta_{jk}} \\
=&\mathbb{E}_\varepsilon\left[\frac{\partial \mathcal{L}_{\mathbb{E}[\text{REC}]}}{\partial \hat{b}_{i}}\right]\Big|_{\hat{a}_{ik}=a_{ik},\hat{b}_i=b_i,\hat{\theta}_{jk}=\theta_{jk}} \\
=&\mathbb{E}_\varepsilon\left[\frac{\partial \mathcal{L}_{\mathbb{E}[\text{REC}]}}{\partial \hat{\theta}_{jk}}\right]\Big|_{\hat{a}_{ik}=a_{ik},\hat{b}_i=b_i,\hat{\theta}_{jk}=\theta_{jk}} \\
=&0 \quad \forall i,j,k
\end{aligned} \tag{2.11}$$

This proves that the true parameters give a local minimum for the expected reconstruction error in the VAE loss.

We now consider the Kullback-Leibler divergence term in the expected loss function. Again assuming independent latent traits, we have

$$\mathcal{L}_{KL} = \mathbb{E}_{q(\theta|u)} \log\left(\frac{q(\hat{\theta}|u)}{p(\theta)}\right) = KL(q(\hat{\theta}|u)||p(\theta)) = -\frac{1}{2}\sum_{k=1}^{K}(1+\log(\hat{\sigma}_k^2)-\hat{\theta}_k^2-\hat{\sigma}_k^2) \tag{2.12}$$

It is clear that this regularization term is minimized (and equal to zero) when $\hat{\theta}_{jk}=0$ and $\hat{\sigma}_{jk}=1$. But what happens when we plug in the "true" student ability values as before? We have

$$\mathcal{L}_{KL}\Big|_{\hat{\theta}=\theta,\hat{\sigma}=\sigma} = KL(p(\theta|u)||p(\theta)) \tag{2.13}$$

Notice that this is the KL divergence between the **true posterior** $p(\theta|u)$ and the **true prior** $p(\theta)$. This is interpreted as the average difference of number of bits required to encode samples of $p(\theta|u)$ using a code optimized for $p(\theta)$, rather than one optimized for $p(\theta|u)$.

We should be okay with accepting this loss, since the true posterior is not actually known, and we are just using the prior as a reference.

2.1.3.2 Requirements on sparsity of Q-matrix

need to look into this idea at some point

TODO: try to show that this is a global minimum for the full VAE loss function. Also take derivatives of the KL loss w.r.t $\hat{\theta}_{jk}$. The Q-matrix may help with an identifiability issue (existence of other local minimums) in solving the system $(a_{ik}\theta_{jk} + b_i)_{jk} = z_i$. The Q-matrix *may* make the solution unique.

2.2 ML2Pvae Software Package for R

The ML2P-VAE method for parameter estimation has been compiled in an easy-to-use software package for R [10]. 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

2.2.1 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 2.1.1. 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 paremeter 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 3
ML2P-VAE RESULTS AND DISCUSSION

The ML2P-VAE method has been used in a various settings in multiple publications [14, 9, 12]. 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 Aritificial 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.

3.1 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 [15, 29, 20].

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 [28]. 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 [33].

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.

3.2 Quantitative Results

3.2.1 Preliminary Results

IJCNN

3.2.2 Ablation Study

This could be helpful

3.2.3 Variational Autoencoder vs Autoencoder

AIED

Shortly after introducing the ML2P-VAE method, comparisons between a variational autoencoder (VAE) and a regular autoencoder (AE) for parameter estimation were made [9]. Recall that Guo et al. proposed a neural network approach to estimating student mastery in 2017 [18]. 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 3.2.1 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 3.1: Statistics for item parameter recovery.

Three error measures for VAE and AE estimates are given in Table 3.1 and Table 3.2. These include absolute value relative bias (AVRB), root mean square error (RMSE) and Pearson correlation (CORR). The statistics for item parameter estimates in Table 3.1, 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 3.1 and Figure 3.2.

Results for student ability parameter estimates are shown in Table 3.2 and Figure 3.3. Again, we see that the error measures from VAE estimates are much

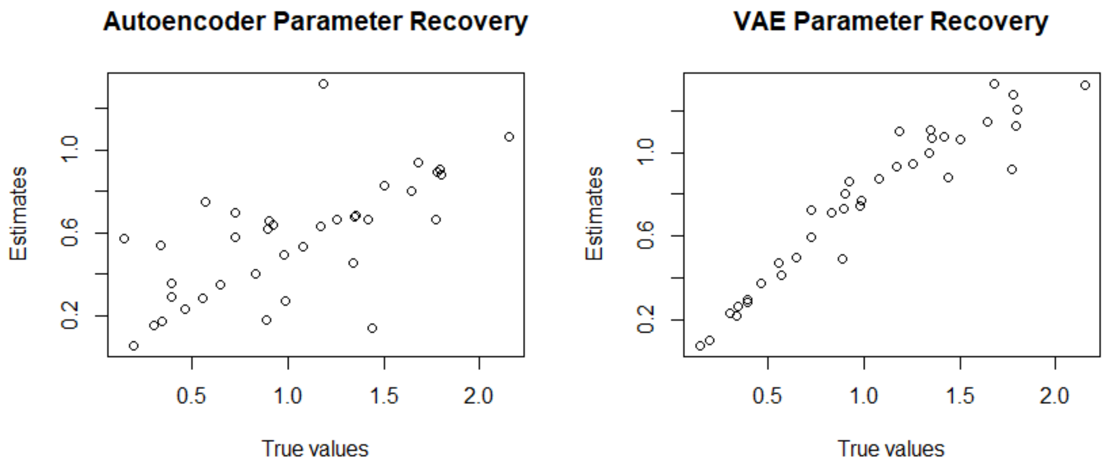

Figure 3.1: Autoencoder and VAE discrimination parameter (a_{ji}) recovery.

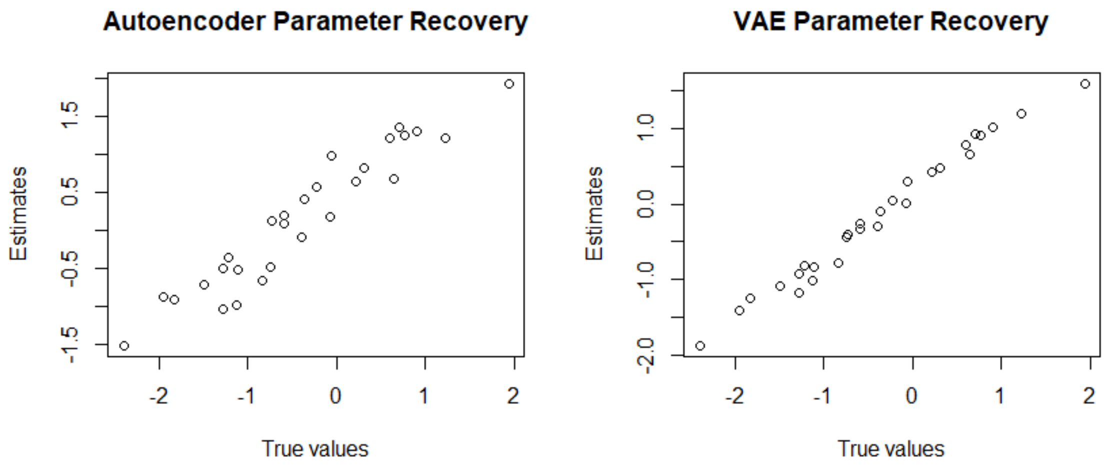

Figure 3.2: Autoencoder and VAE difficulty parameter (b_i) recovery.

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 3.3 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 3.2: 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 3.1. The ML2P model can suffer from an identifiability issue without the assumption that student ability parameters follow some probability distribution [19]. 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 incorporate an extra piece of domain knowledge in the prior distribution of Θ, leading to more accurate estimates.

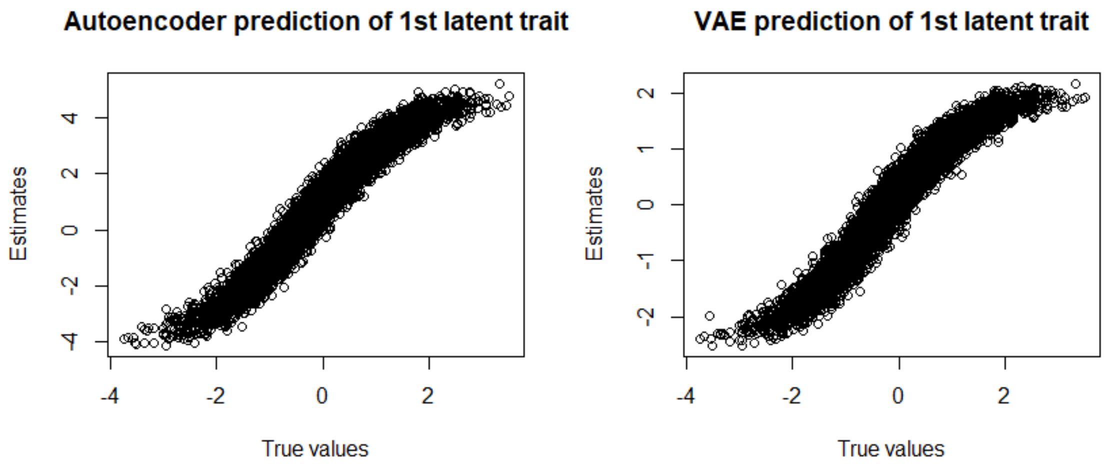

Figure 3.3: Autoencoder and VAE predictions for θ_1.

3.2.4 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 2.1.2. These are compared against Metropolis-Hastings Robbins-Monro (MHRM) [7], Quasi Monte-Carlo Expectation Maximization (QMCEM) [8], and Monte-Carlo Expectation Maximization (MCEM) [5]. This work has been submitted to the Machine Learning journal [12].

update if this ever gets accepted

A summary of each method's performance is given in Table 3.3. 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 [8]. 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 regularization parameter in Equation **??** was fixed at $\lambda = 1$. 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 sigmoidal 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.

fix reference

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 dimension 6-20), the longer runtime is more due to the number of data samples, rather

Data Set	Method	a.RMSE	a.BIAS	a.COR	b.RMSE	b.BIAS	b.COR	θ.RMSE	θ.BIAS	θ.COR	Runtime
	MHRM	0.0693	0.0319	0.9986	0.0256	-0.0021	0.9999	0.714	-0.0033	0.7006	1110s
(i)	QMCEM	0.149	-0.067	0.9939	0.0376	-0.002	0.9998	0.7206	0.0023	0.6939	322s
6 abilities	MCEM	0.1497	-0.0633	0.9936	0.0383	0.0035	0.9997	0.7206	-0.0016	0.6938	1009s
Sim-6	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
	MHRM*	0*	0*	1*	0*	0*	1*	0*	0*	1*	162s
(ii)	QMCEM	0.0159	0.0035	0.9999	0.0067	-0.0005	1	0.0111	0.0007	0.9999	192s
3 abilities	MCEM	0.0228	0.0148	0.9998	0.0064	-0.0008	1	0.0132	0.0026	0.9998	33s
ECPE	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
	MHRM	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
(iii)	QMCEM	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
20 abilities	MCEM	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Sim-20	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 3.3: 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

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.

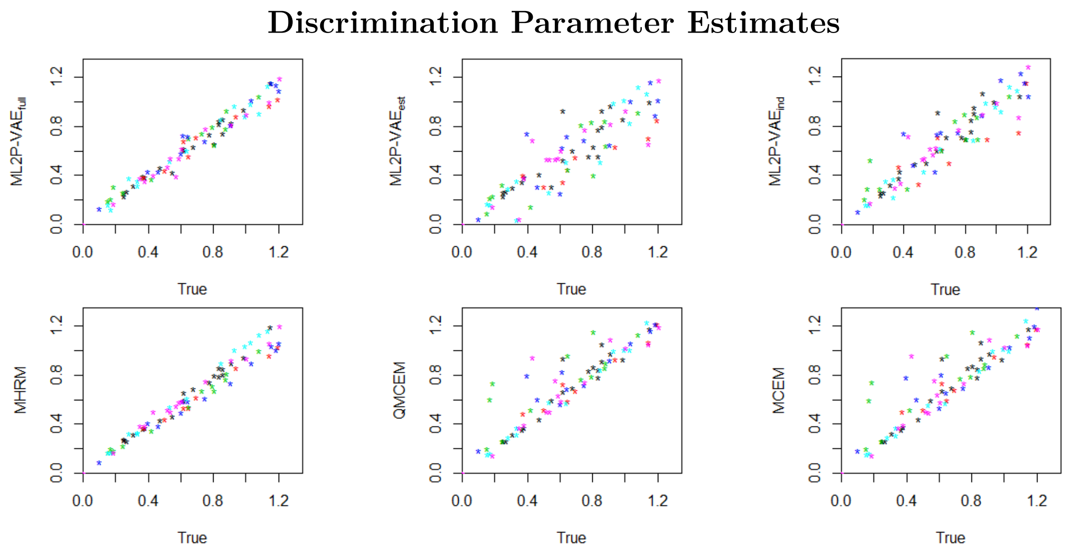

Figure 3.4: Correlation plots of discrimination parameter estimates for the Sim-6 dataset with 50 items and 6 latent traits. ML2P-VAE estimates are on the top row, and traditional method estimates are on the bottom row

Some of the results are visualized in Figures 3.4, 3.5, and 3.6 for Sim-6, ECPE, and Sim-20, respectively. Figure 3.4 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.

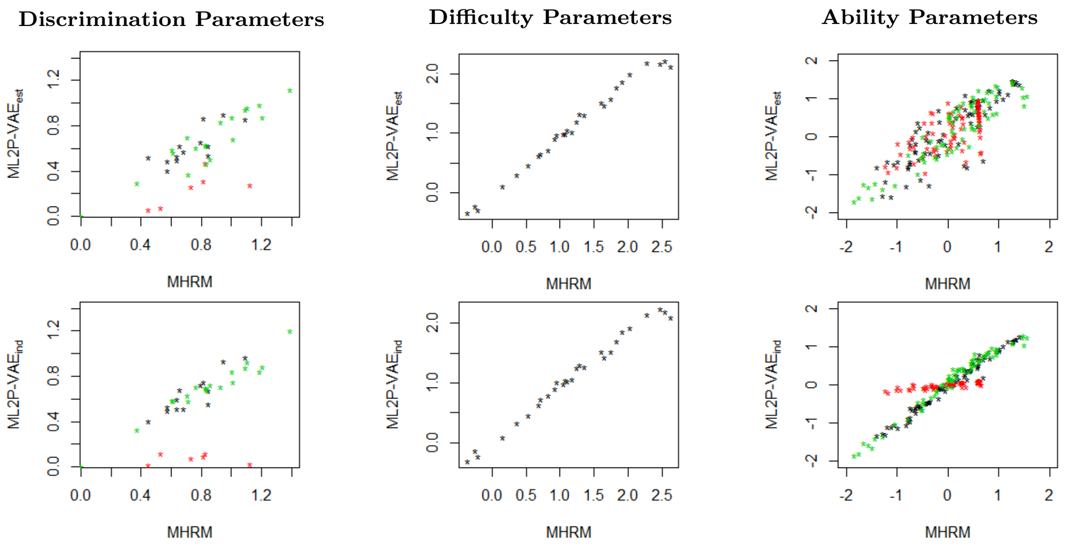

Figure 3.5: Estimates from ML2P-VAE methods plotted against "accepted" MHRM estimates from the ECPE dataset

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 3.3, 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.

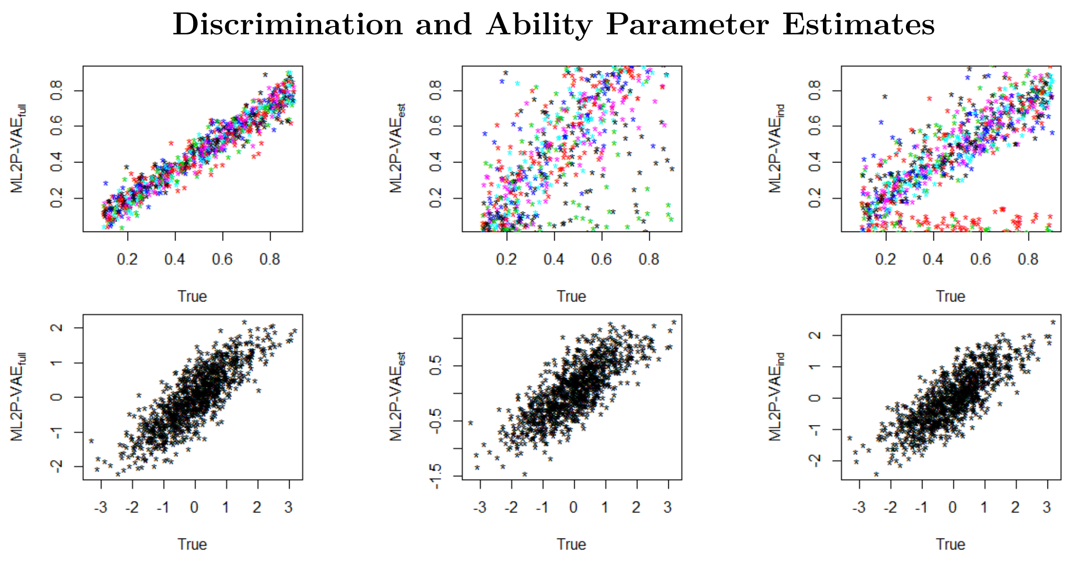

Figure 3.6: ML2P-VAE parameter estimates for Sim-20 with 200 items and 20 latent traits. The top row shows discrimination parameter correlation, and the bottom row shows ability parameter correlation

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.

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 3.3 for θ in Sim-20 are comparable to those of Sim-6. The plots in Figure 3.6 show this high correlation in all three ML2P-VAE variants.

3.2.4.1 Effect of Training Data Size

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

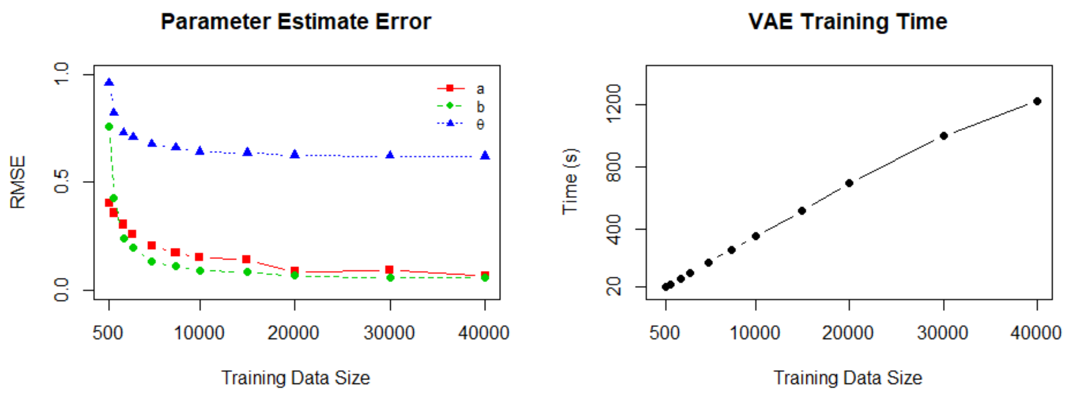

Figure 3.7: Performance of ML2P-VAE$_{full}$ on data set (iii) when trained on data sets of increasing size. The left plot gives the test RMSE after using different sizes of training data, and the right plot shows the time required to train the neural network

The longer runtimes in Table 3.3 for Sim-20 can be attributed more to the fact that there were 50,000 data samples, rather than the large latent dimension. The left plot of Figure 3.7 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 3.7 shows that training time grows linearly with the size of training data.

Both plots in Figure 3.7 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.

could add in some stuff from ML paper Discussion

am I going to do anything with 3PL?

CHAPTER 4
KNOWLEDGE TRACING BACKGROUND

Knowledge Tracing (KT) is a task introduced by Corbett and Anderson in 1995 [13]. 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

$$P(L_n) = P(L_{n-1}|\text{evidence}) + (1 - P(L_{n-1}|\text{evidence})) \cdot P(T). \tag{4.1}$$

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 the 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 4.1, 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 5, 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.

4.1 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

$$P(c_{t+1} = 1 | (q_0, c_0), (q_1, c_1), \ldots, (q_t, c_t), (q_{t+1}, ?)). \tag{4.2}$$

Most neural networks optimize the predicted probability in Equation 4.2 by minimizing the cross-entropy loss function, as described in Equation 1.9.

4.1.1 Deep Knowledge Tracing

In 2015, the first use of neural networks for knowledge tracing was introduced by Piech et al. [25]. 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.

Do I need to give background on RNN and LSTM?

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 feedforward 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.

4.1.2 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) [35]. 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 [31].

Next, *read* from the value matrix by computing

$$r_t = \sum_{i=1}^{K} w_t(i) M^v(i). \tag{4.3}$$

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

$$M_t^v(i) = (M_{t-1}^v(i)[1 - w_t(i)e_t]) + w_t(i)a_t \tag{4.4}$$

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.

should I include image of DKVMN architecture?

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.

4.1.2.1 Deep-IRT

Deep-IRT, proposed by Chun-Kit Yeung [34] 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:

$$p_t = \frac{1}{1 + \exp\left(\beta_k - 3 \cdot \theta_{tk}\right)} \tag{4.5}$$

These modifications provide a link to the Rasch model in Equation 1.3. The multiplication by 3 is for practical reasons to re-scale θ_{tk}. However, note that in Equation 4.5, 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.

4.1.3 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 [31], which rely on calculating the correlation between pairs of words in a sentence. Popular models such as BERT [16] and GPT-3 [6] 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 [24].

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 ty a weights matrix to obtain a *query* vector $\vec{q_t} = W^Q e_t$. The interaction embedding m_i is used to create two vectors: a *key* vector $\vec{k_i} = W^K m_i$ and a *value* vector $\vec{v_i} = W^V m_i$.

The general idea is that $\vec{k_i}$ serves as the identifier of a past interaction, and

$\vec{q_t}$ 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 $\vec{v_i}$ holds more abstract information about the corresponding interaction. The keys and values are organized into matrices K and V. We calculate the attention

$$a_t = \text{softmax}\left(\frac{K\vec{q_t}}{\sqrt{d}}\right)V \tag{4.6}$$

The value $\frac{K\vec{q_t}}{\sqrt{d}}$ yields a vector where each entry is the dot product between the current exercise query $\vec{q_t}$ and an interaction key $\vec{k_i}$. This is scaled by dimension and softmaxed, resulting in a weighted sum of the value vectors $\vec{v_i}$.

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.

4.1.4 Performance Factors Analysis

*** Not sure if I will include Deep-PFA. If not, then don't need to talk much about PFA.

CHAPTER 5
DEEP, INTERPRETABLE METHODS FOR KNOWLEDGE TRACING

5.1 Deep Performance Factors Analysis

*** May not include anything about Deep PFA

5.2 Incorporating IRT into Knowledge Tracing

*** wait for reviews from AIED paper

CHAPTER 6
COMPARISON OF KNOWLEDGE TRACING METHODS

*** wait for reviews from AIED

6.1 Data Description

Describe each dataset used here.

6.2 Experiment Details

Hyper parameters here

6.3 Results

6.3.1 Deep PFA

6.3.2 IRT parameter recovery + KT

CHAPTER 7
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.

7.1 Health Sciences

Beck Depression Inventory

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

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

TODO: citation.

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.

7.2.1 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. [11].

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.

7.2.2 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 performance 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 7.1 [11]. 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.

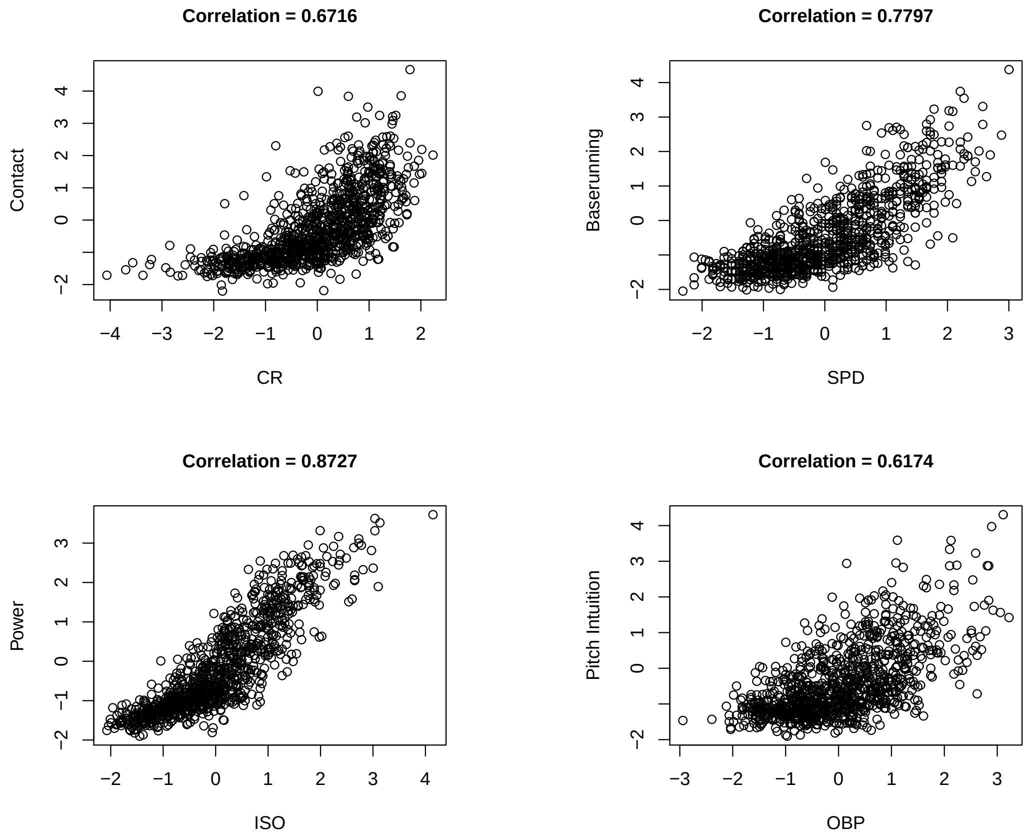

Figure 7.1: Each latent skill's estimates plotted against its evaluation statistic.

REFERENCES

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

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

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

[4] 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.

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

[6] 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.

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

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

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

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

[11] 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.

[12] 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*, Under review 2020.

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

[14] 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.

[15] 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.

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

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

[18] 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.

[19] 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.

[20] 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.

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

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

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

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

[25] 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.

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

[27] 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.

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

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

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

[31] 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.

[32] 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.

[33] 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.

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

[35] 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.

[36] 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.