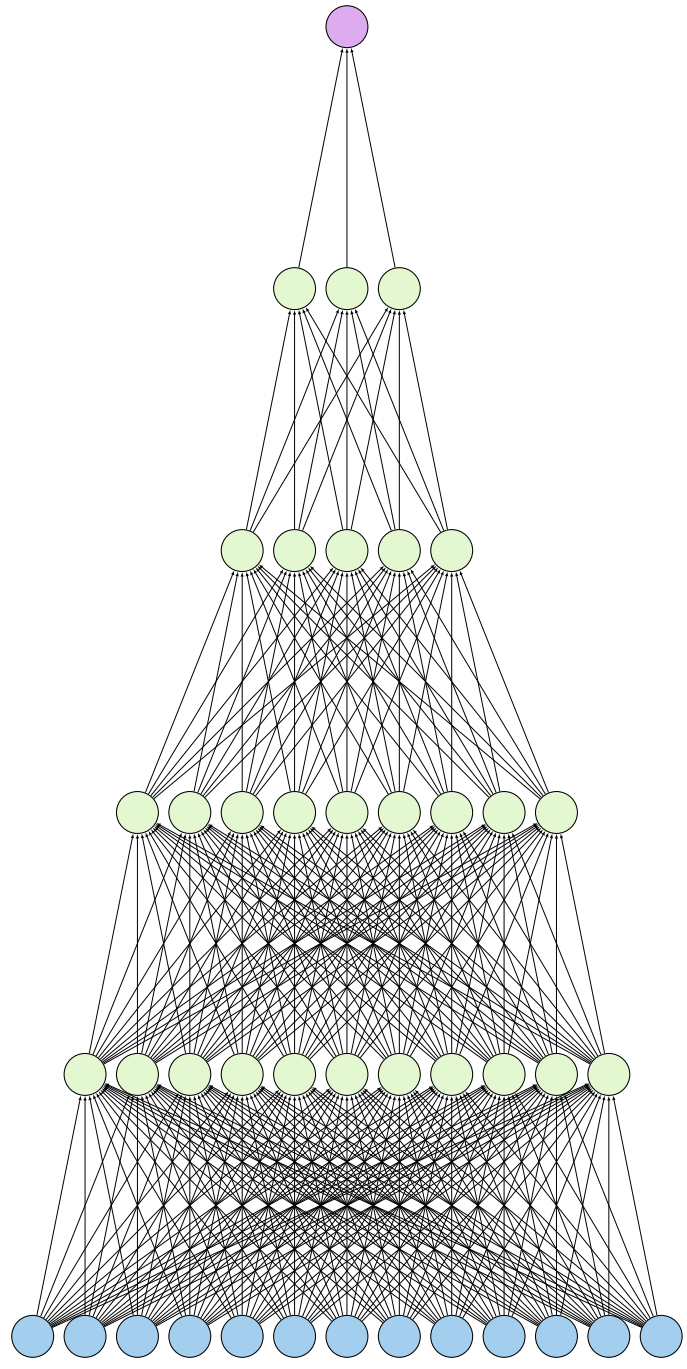
Foundations of Deep Learning for the Social Sciences

Christopher Urban

University of North Carolina at Chapel Hill



Review of Today's Learning Outcomes

You will learn:

- How to fit structural equation models using backpropagation and stochastic gradient-based optimization
- The defintion of the autoencoder and applications
- The basics of non-amortized and amortized variational inference
- How to use variational methods to fit complex latent factor models

Why Latent Variable Models?

Social scientists want to reduce the dimensionality of data to make it easier to understand.

In my field (psychology), researchers often use questionnaires to indirectly measure variables like

- abilities;
- personality characteristics; and
- mental illnesses.

	Disagree		Neutral		Agree
I am the life of the party.	○	○	○	○	○
I feel little concern for others.	○	○	○	○	○
I am always prepared.	○	○	○	○	○
I get stressed out easily.	○	○	○	○	○
I have a rich vocabulary.	○	○	○	○	○
I don't talk a lot.	○	○	○	○	○

Some categorical questions from an online Big-Five Personality Factors questionnaire.

Condensing into latent constructs facilitates understanding.

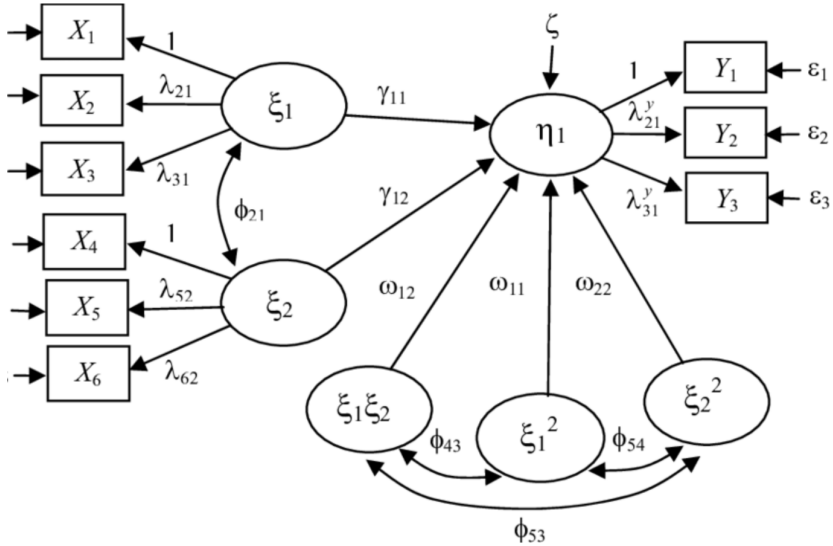
Framework 1: Structural Equation Modeling

The basic idea is that we have two parts:

1. a *measurement model* that describes how we're using the observed variables to measure the latent constructs; and
2. a *structural model* that describes the relationships between the latent constructs.

Our goal is for the model to accurately reproduce the covariance matrix of the observed data.

Can test a variety of complicated hypotheses (often shown as schematic diagrams).



Example 1: Linear Factor Analysis

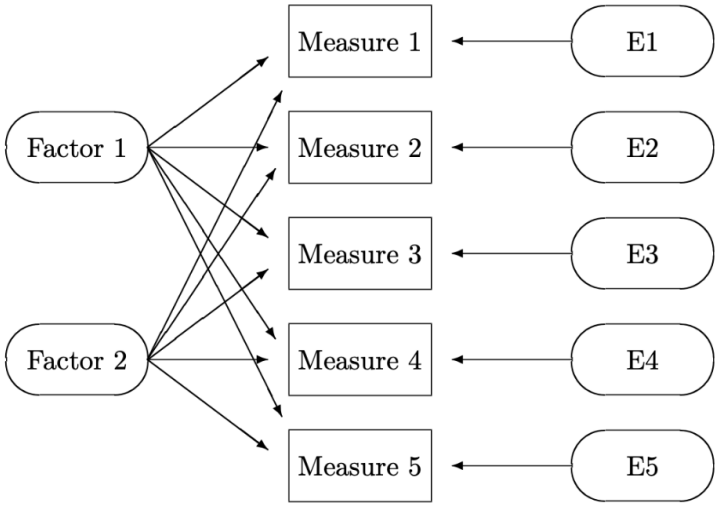
We assume the continuous observed variables $\mathbf{y}$ are generated by a smaller set of continuous latent variables $\mathbf{x}$:

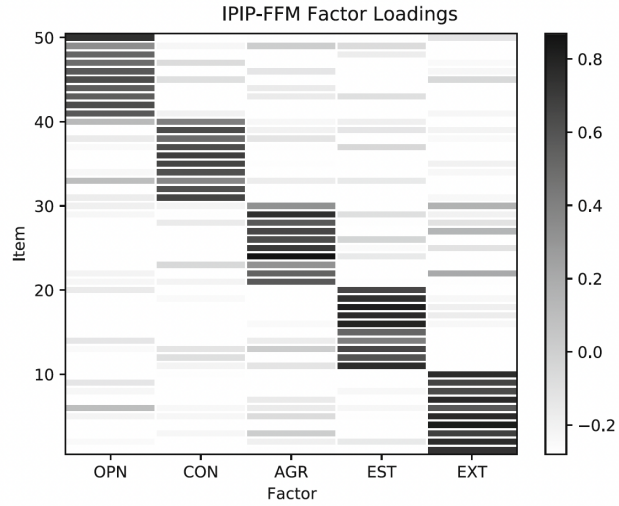
$$\mathbf{y} = \mathbf{\Lambda x} + \varepsilon,$$

where $\mathbf{\Lambda}$ is the *loadings matrix* and ε is a vector of residuals.

Similar to linear regression, but now the predictors are latent.

The loadings matrix is a tool for interpreting the factors!

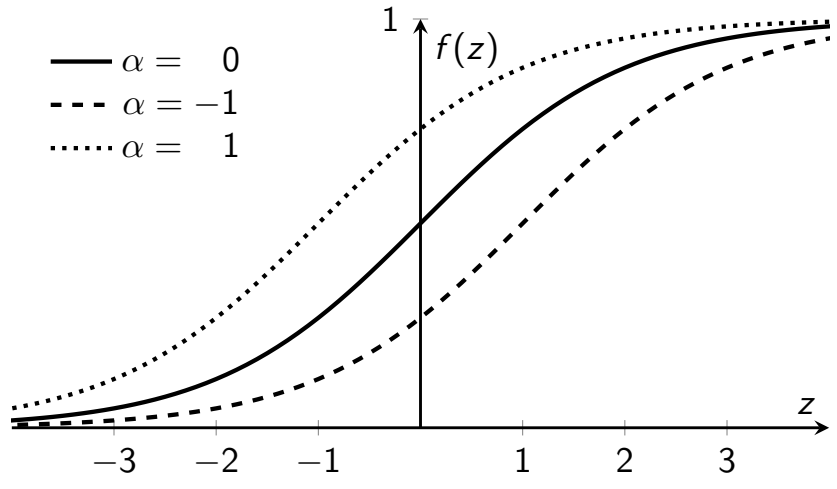




Framework 2: Item Response Theory

Item response theory (IRT) is a family of models for categorical observed data with underlying continuous latent variables.

IRT provides useful information about question properties like difficulty (i.e., how hard is this question?) and discrimination (i.e., how easily can we tell respondents apart?).



Item response curves with different difficulty parameters.

Fitting IRT models is computationally challenging.

Example 2: Multidimensional 2-Parameter Logistic Model

The M2PL is a model for binary observed data.

The M2PL says that the observed data are generated as follows:

$$y_j \mid \mathbf{x} \sim \text{Bernoulli}(y_j \mid \sigma(\boldsymbol{\beta}^\top \mathbf{x} + \alpha_j)),$$

where y_j is the response to item j, $\boldsymbol{\beta}$ is a loadings vector, and α_j is an intercept for item j.

This is just a logistic regression of the items on the latent factors.

Similar to linear factor analysis, we interpret the factors using the loadings matrix.

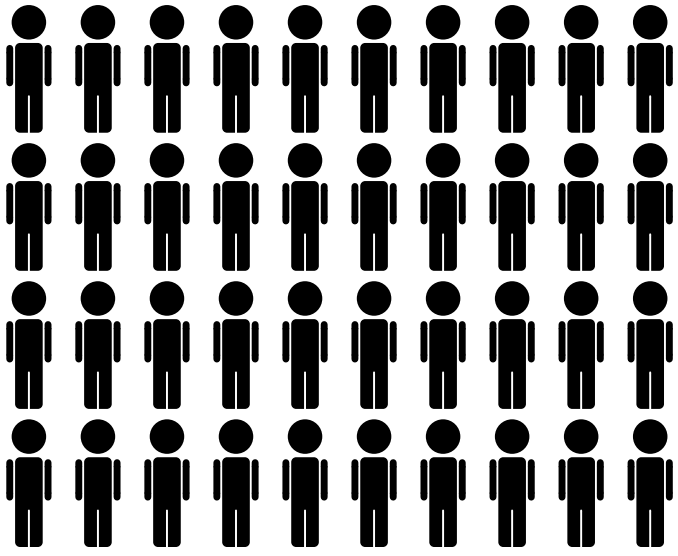
Shortcomings of the Traditional Approaches

Over the past decade, lots of large-scale item response data has been (digitally) collected:

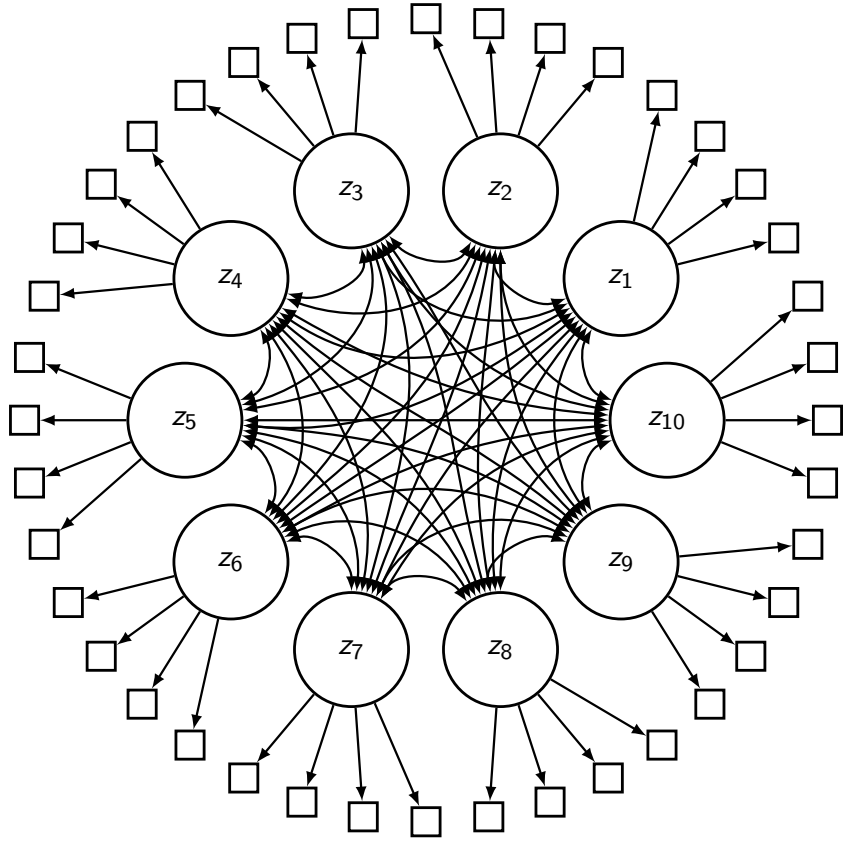
- Assessment data from e-learning technologies
- Mental health symptom data from surveys and clinics
- Customer preference data from retailers and streaming services
- Personality data from online inventories

Traditional latent variable models aren't always fast or flexible enough to model these kinds of data!

Computational Challenges with Modern Data Sets



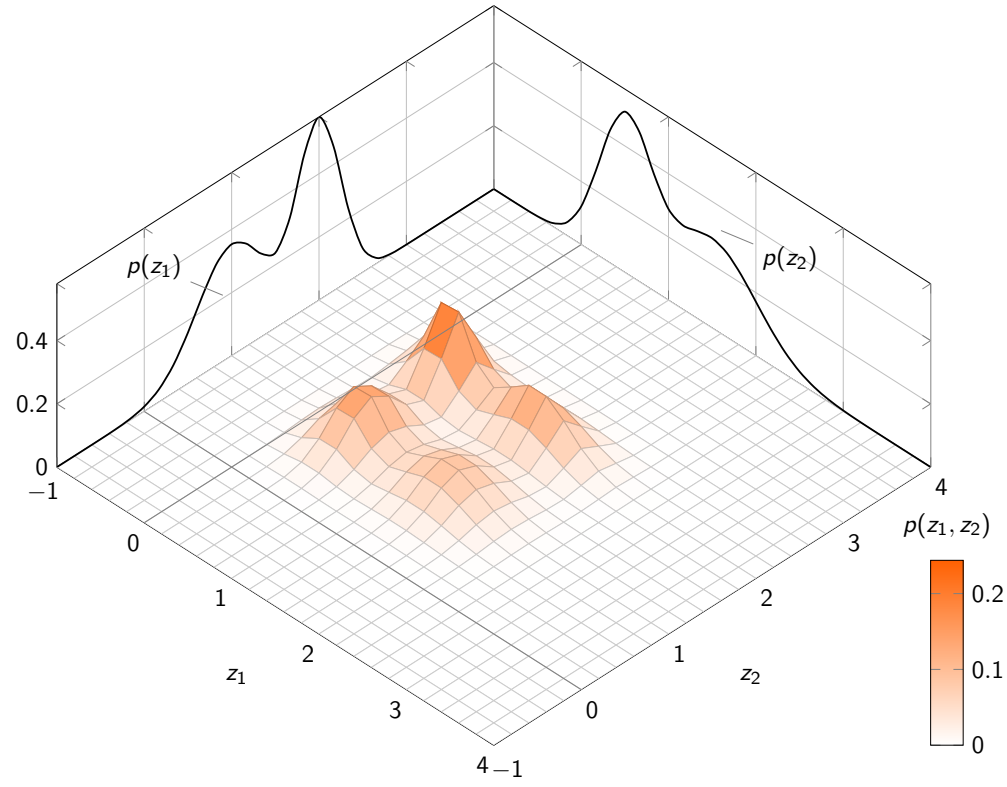
Lots of people



Lots of latent and observed variables

Modeling Challenges with Modern Data Sets

- Non-normal latent variables
- Non-linear associations between LVs and OVs
- Missing data



Solution: Deep Learning

The deep learning tools we learned about yesterday provide building blocks to solve a few of these challenges.

- Lots of people and observed variables?
  $\rightarrow$ Backpropagation and stochastic gradient methods!
- Non-linear associations between variables?
  $\rightarrow$ Neural networks!

We'll learn about some new deep learning tools today that let us handle latent variable models with complicated log-likelihoods.

Review of Estimation for Structural Equation Models

The model has two parts:

$$\begin{aligned} \mathbf{y} &= \boldsymbol{\Lambda}\mathbf{x} + \boldsymbol{\varepsilon} && \text{Measurement model} \\ \mathbf{x} &= \mathbf{B}_0\mathbf{x} + \boldsymbol{\xi} && \text{Structural model} \end{aligned}$$

All the estimable parameters can be collected in a big vector:

$$\boldsymbol{\delta} = (\text{vec}(\boldsymbol{\Lambda})^\top, \text{vech}(\boldsymbol{\Theta})^\top, \text{vech}(\boldsymbol{\Psi})^\top, \text{vec}(\mathbf{B}_0)^\top)^\top,$$

with $\boldsymbol{\Theta}$ the covariance matrix of $\boldsymbol{\varepsilon}$ and $\boldsymbol{\Psi}$ the covariance matrix of $\mathbf{x}$.

Letting $\boldsymbol{\delta} = \boldsymbol{\delta}(\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ are the free parameters, the model-implied covariance matrix is:

$$\boldsymbol{\Sigma}(\boldsymbol{\theta}) = \boldsymbol{\Lambda}(\mathbf{I} - \mathbf{B}_0)^{-1}\boldsymbol{\Psi}(\mathbf{I} - \mathbf{B}_0)^{-\top}\boldsymbol{\Lambda}^\top + \boldsymbol{\Theta}.$$

We want this to look like observed covariance matrix $\mathbf{S}$.

SEM Fitting Functions

Our SEM loss function should measure the discrepancy between $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ and $\mathbf{S}$.

Then, when we minimize it, we're choosing the $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ that best fits the observed covariance structure.

One example is the *maximum likelihood fitting function*, which follows from the assumption that $\mathbf{y} \sim \mathcal{N}(\mathbf{y} \mid \mathbf{0}, \boldsymbol{\Sigma}(\boldsymbol{\theta}))$:

$$F_{\text{ML}}(\boldsymbol{\theta}) = \log|\boldsymbol{\Sigma}(\boldsymbol{\theta})| + \text{tr}\left[\mathbf{S}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\theta})\right].$$

We can also use different fitting functions, such as the *generalized least squares* (GLS) function:

$$F_{\text{GLS}}(\boldsymbol{\theta}) = [\text{vech}(\mathbf{S}) - \text{vech}(\boldsymbol{\Sigma}(\boldsymbol{\theta}))]^\top \mathbf{W} [\text{vech}(\mathbf{S}) - \text{vech}(\boldsymbol{\Sigma}(\boldsymbol{\theta}))],$$

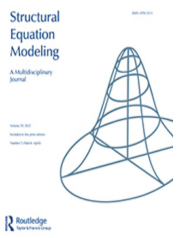
where different choices for the weight matrix $\mathbf{W}$ produce different estimators.

SEM and Deep Learning-Based Optimization

It's *very* straightforward to combine SEM with the deep learning concepts we've already learned.

If we compute the fitting function using a deep learning framework with automatic differentiation (e.g., PyTorch, Tensorflow, etc.), we can backpropagate and optimize the fitting function using stochastic gradient methods.

It's also really easy to add extensions like regularization.



Structural Equation Modeling: A Multidisciplinary Journal

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/hsem20

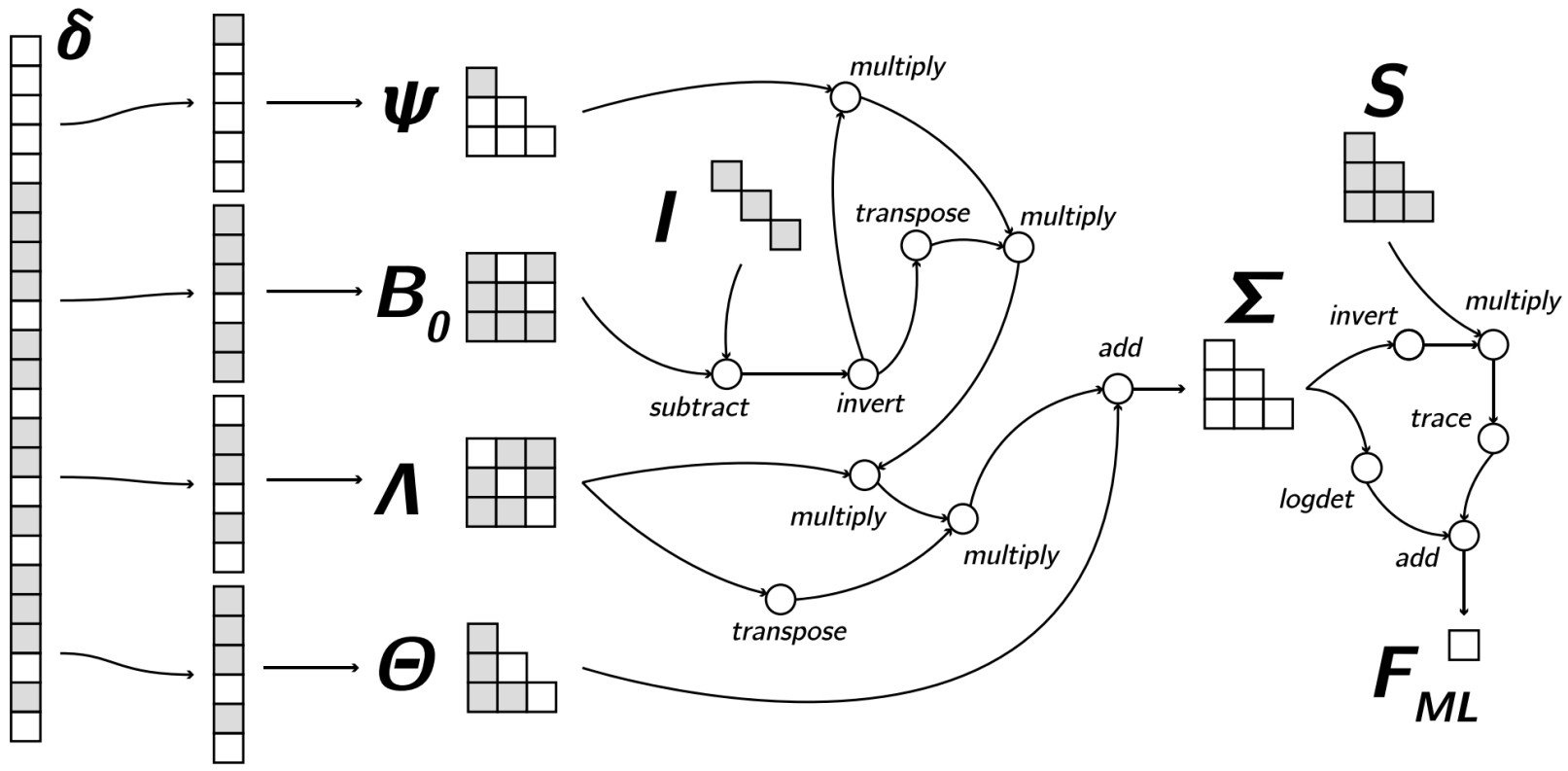
Flexible Extensions to Structural Equation Models Using Computation Graphs

Erik–Jan van Kesteren & Daniel L. Oberski

To cite this article: Erik–Jan van Kesteren & Daniel L. Oberski (2022) Flexible Extensions to Structural Equation Models Using Computation Graphs, Structural Equation Modeling: A Multidisciplinary Journal, 29:2, 233-247, DOI: 10.1080/10705511.2021.1971527

To link to this article: https://doi.org/10.1080/10705511.2021.1971527

Here's a pretty picture of the computation graph for $F_{\mathrm{ML}}(\boldsymbol{\theta})$.



Learning Outcomes Checkpoint

- We've discussed fitting SEMs using backpropagation and stochastic gradient-based optimization.
- Now we'll move on to fitting more challenging latent variable models, beginning with an introduction to autoencoders.

Review of Estimation for Item Response Theory Models

Fitting IRT models with a lot of factors is computationally intensive.

General latent factor models (including IRT models) look like this:

$$\begin{aligned} \mathbf{x} &\sim p_{\boldsymbol{\xi}}(\mathbf{x}) && \text{latent variable prior} \\ \mathbf{y} \mid \mathbf{x} &\sim p_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x}) && \text{measurement model} \end{aligned}$$

The marginal log-likelihood of the observed data looks like this:

$$\log \mathcal{L}(\boldsymbol{\delta} \mid \mathbf{Y}) = \sum_{i=1}^{N} \left[\int p_{\boldsymbol{\theta}}(\mathbf{y}_i \mid \mathbf{x}) p_{\boldsymbol{\xi}}(\mathbf{x}) d\mathbf{x} \right],$$

where $\boldsymbol{\delta} = (\boldsymbol{\theta}^\top, \boldsymbol{\xi}^\top)^\top$ collects all unknown parameters.

There's *no analytic simplification*. We need to approximate the N integrals over $\mathbb{R}^D$ numerically!

IRT and Deep Learning-Based Optimization

Fitting IRT models in deep learning frameworks is less straightforward than for SEM as we no longer have a nice closed-form log-likelihood.

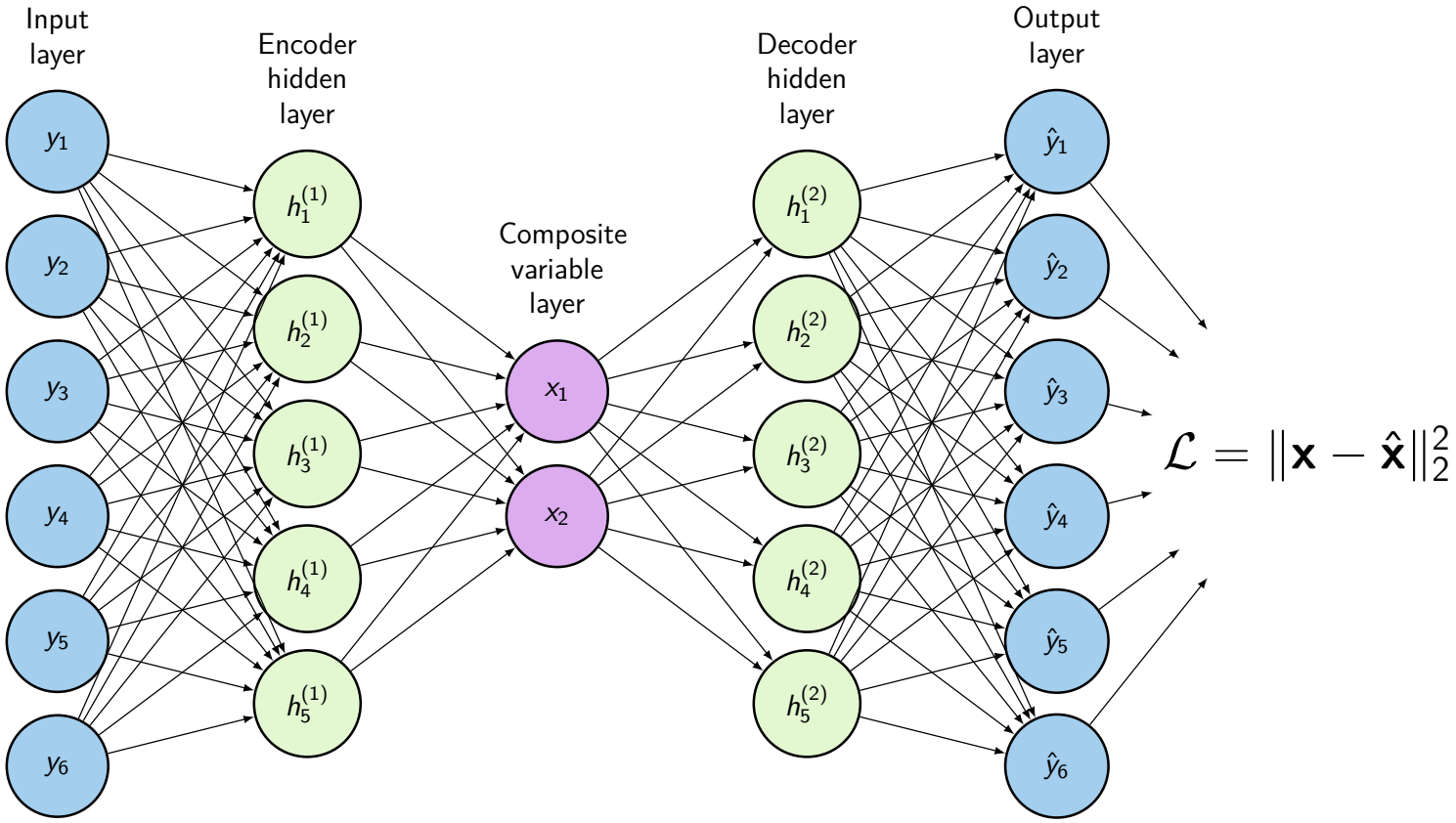
To fit IRT models, we'll introduce some new tools from deep learning-based approximate inference to approximate the marginal log-likelihood.

Working in the approximate inference framework will also let us introduce flexible extensions of IRT models — e.g., models with neural net-based measurement models and with flexible latent prior distributions.

We'll start by introducing *autoencoders*.

Autoencoders

Autoencoders (AEs) are neural nets that copy their input to their output.



Autoencoder Uses

AEs are primarily used for dimensionality reduction.

- Linear AE weights span the same subspace as PCA loadings (Baldi and Hornik, 1988).
- Full AEs perform a nonlinear generalization of PCA.

This reduces the size of predictive models applied to the data and is useful for clustering.

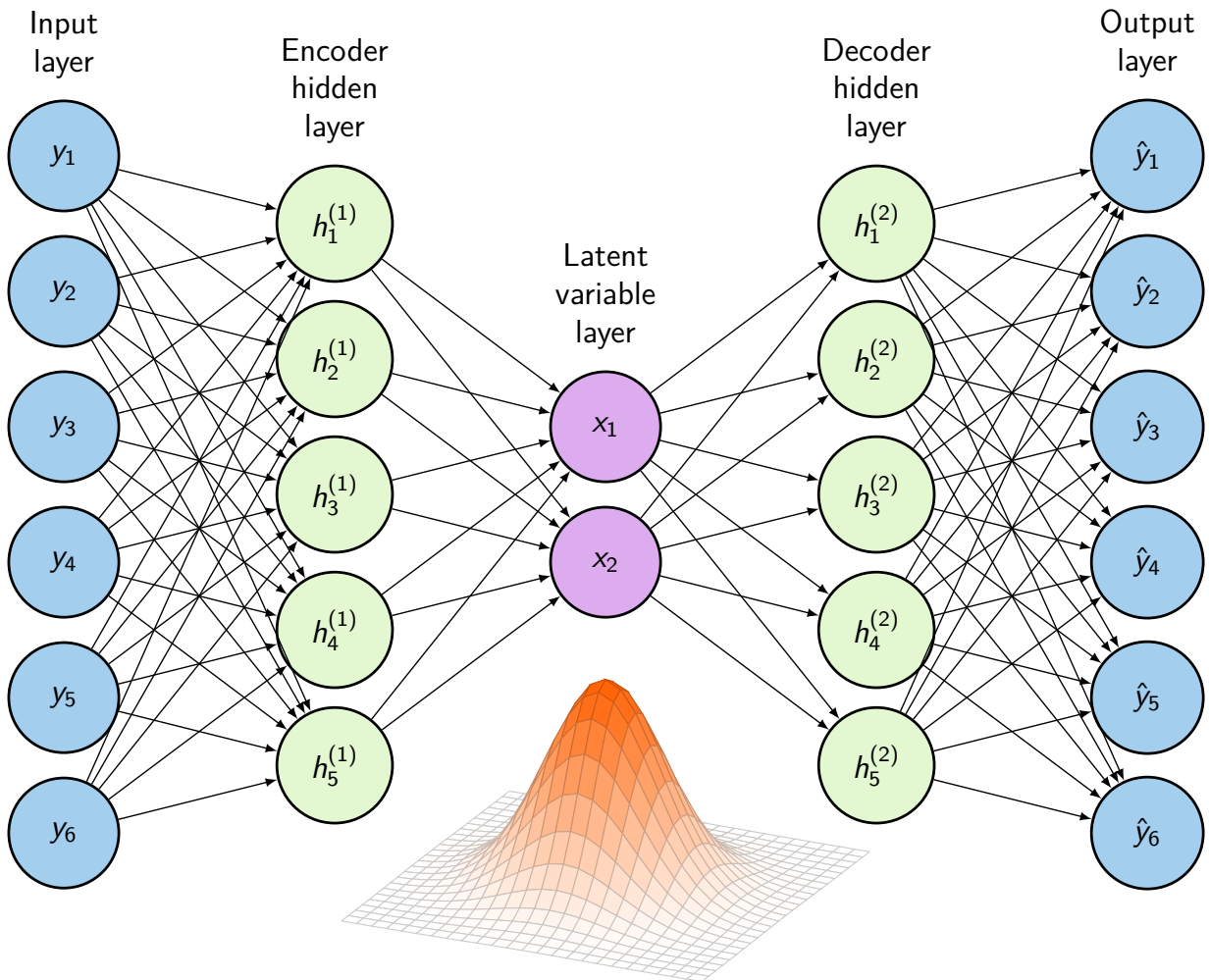
Unfortunately, in their most basic form, AEs are not generative models – that is, they don't model the data generating process, and therefore can't simulated realistic fake data.

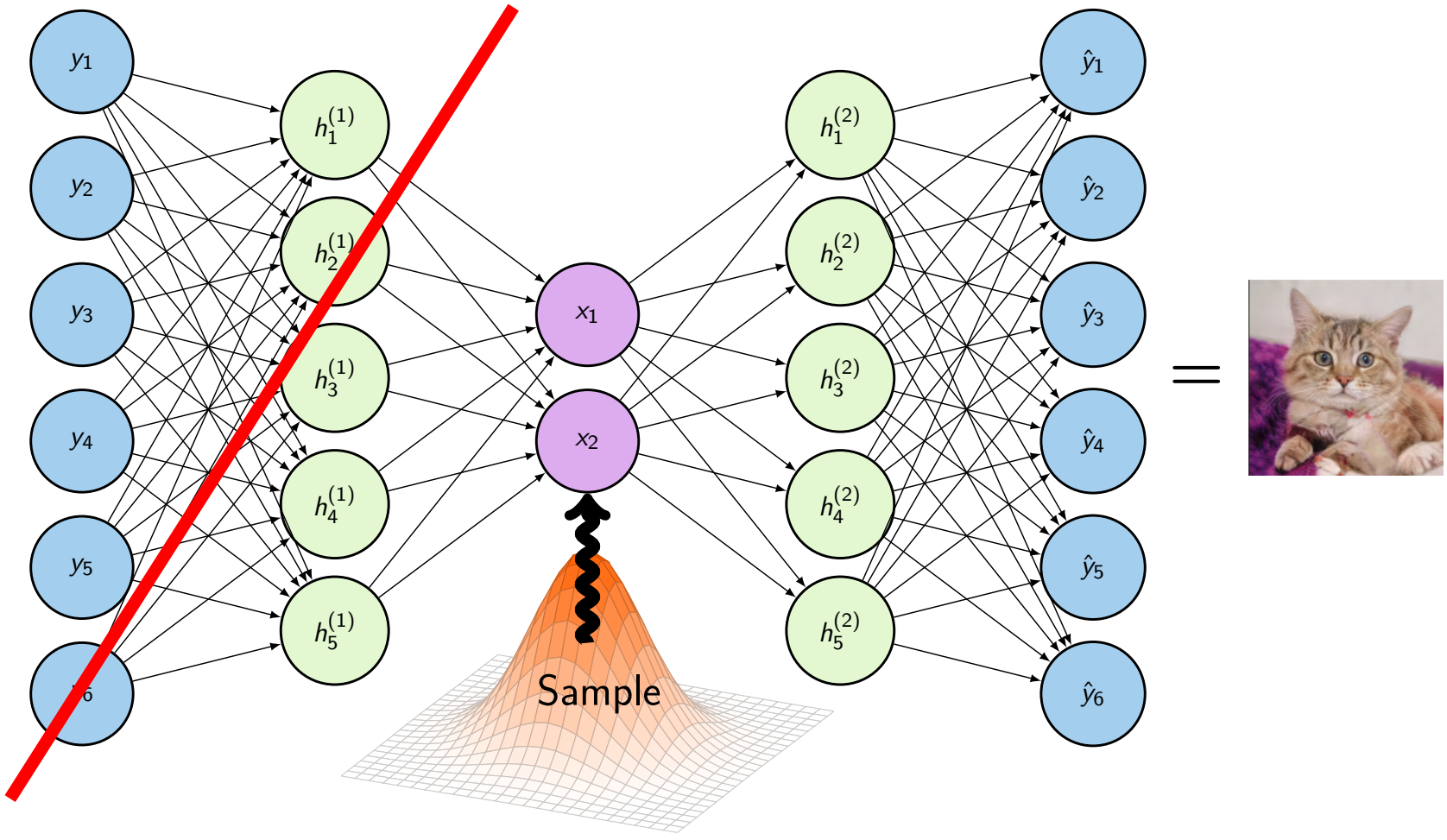
Learning Outcomes Checkpoint

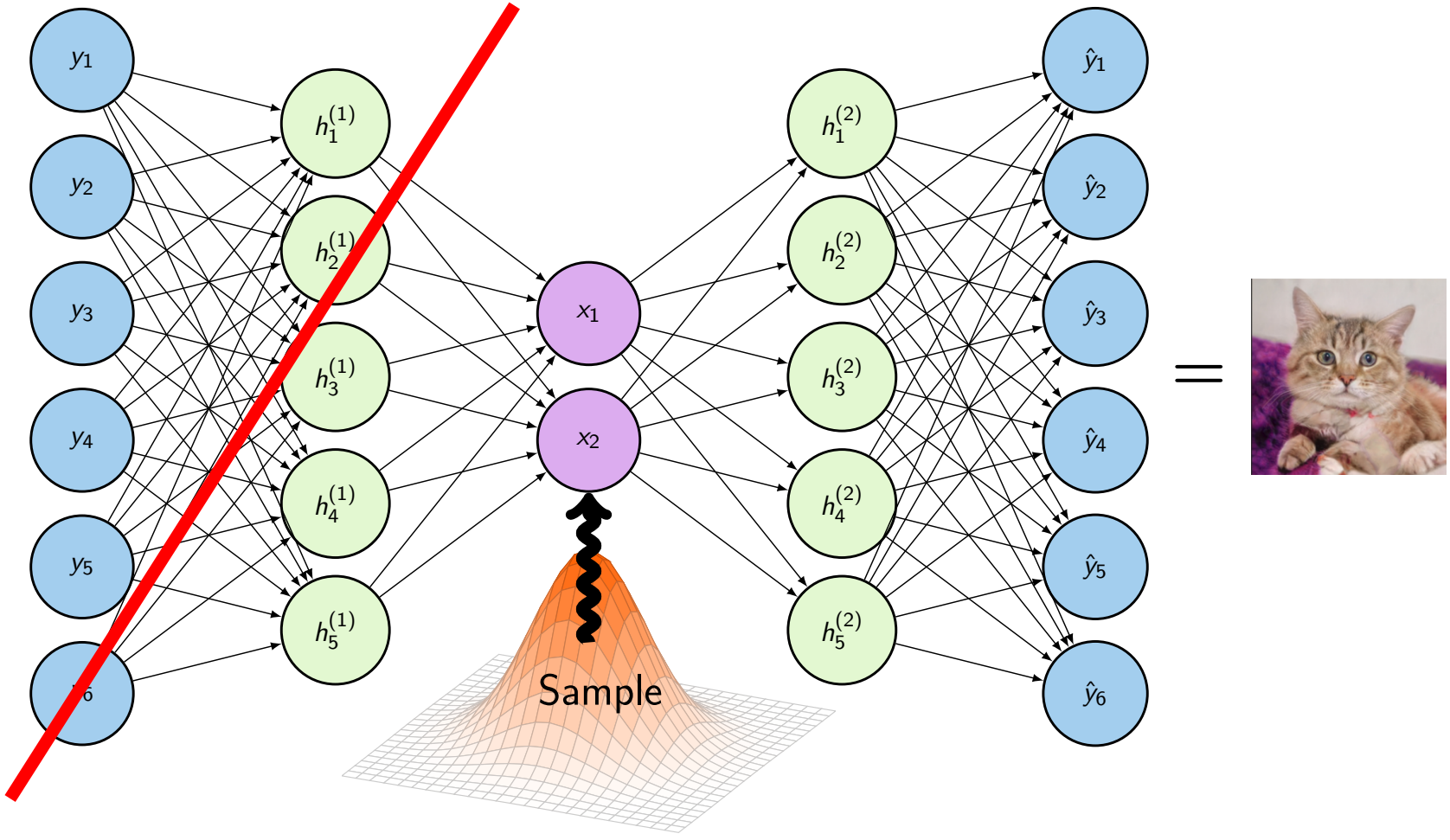
- We've discussed why fitting IRT models is hard and introduced autoencoders.
- We'll next get into the intuition behind variational autoencoders, and explain the difference between amortized and non-amortized variational inference.

Variational Autoencoders — Intuition

Similar idea as autoencoder, except the bottleneck has latent variables with a distribution we choose.




y_1
y_2
y_3
y_4
y_5
y_6
$h_1^{(1)}$
$h_2^{(1)}$
$h_3^{(1)}$
$h_4^{(1)}$
$h_5^{(1)}$
x_1
x_2
Sample
$h_1^{(2)}$
$h_2^{(2)}$
$h_3^{(2)}$
$h_4^{(2)}$
$h_5^{(2)}$
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
$\hat{y}_4$
$\hat{y}_5$
$\hat{y}_6$
=



This is just nonlinear factor analysis:

$$\mathbf{x} = g(\mathbf{z}) + \varepsilon,$$

where $g(\cdot)$ is approximated by a neural net.

The Connection Between VAE and Psychometrics

What if we change $g(\cdot)$ to be something other than a neural net?

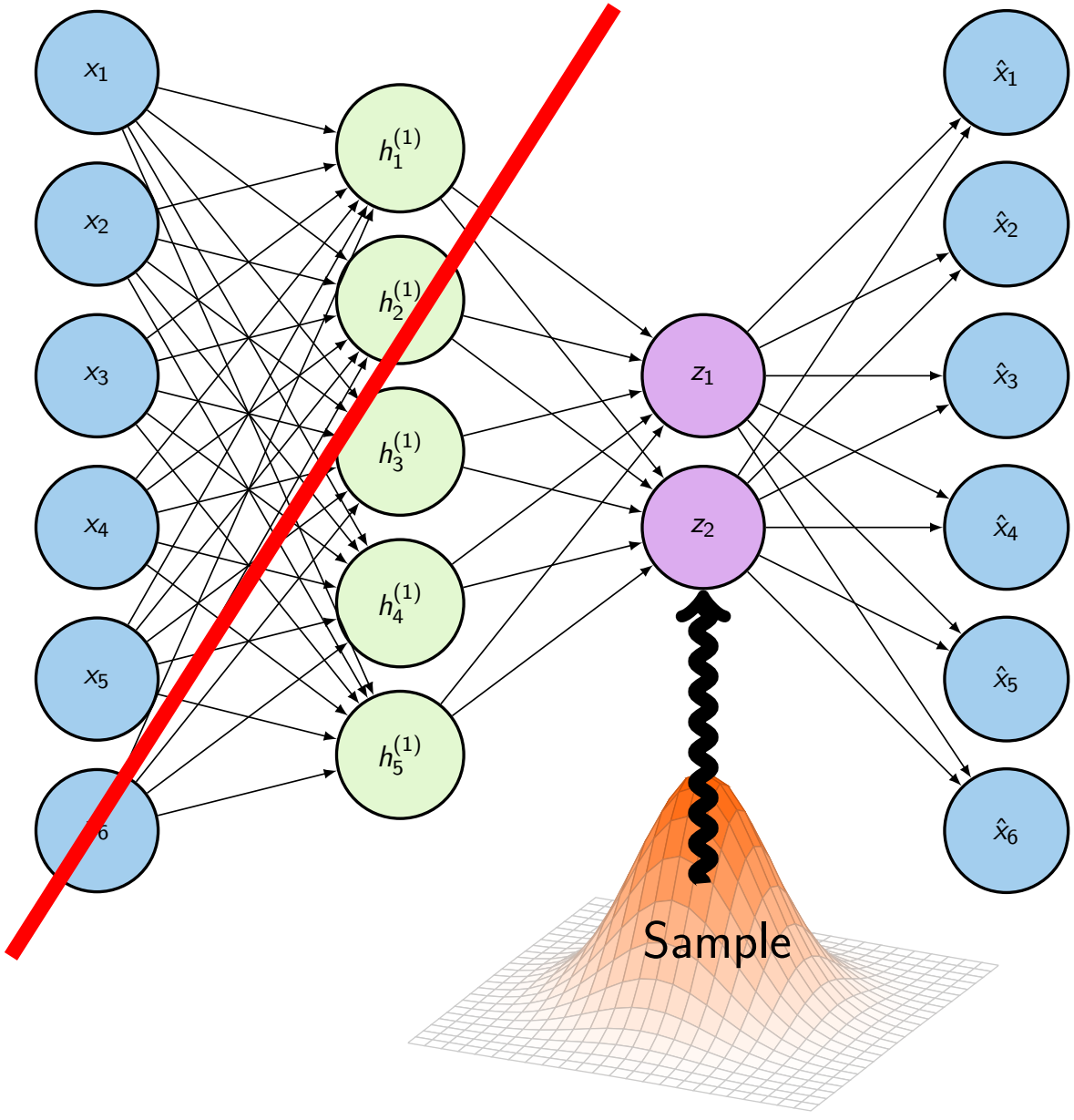
If $g(\cdot)$ is linear, we get linear factor analysis:

$$\begin{aligned}\mathbf{y} &= g(\mathbf{x}) + \boldsymbol{\varepsilon} \\ &= \boldsymbol{\Lambda}\mathbf{x} \ \ + \boldsymbol{\epsilon}.\end{aligned}$$

If $g(\cdot)$ is linear with an inverse logistic link, we get IRT's M2PL:

$$\begin{aligned}\mathbf{p} &= g(\mathbf{x}) = \sigma(\mathbf{B}\mathbf{x} + \boldsymbol{\alpha}) \\ p_\theta(\mathbf{y} \mid \mathbf{x}) &= \text{Bernoulli}(\mathbf{y} \mid \mathbf{p}).\end{aligned}$$

The right side looks a lot like the latent variable models we're used to!



What Makes VAEs Variational?

The VAE defines the following generative model:

$$\begin{aligned} \mathbf{x} &\sim p_{\xi}(\mathbf{x}) && \text{latent variable prior} \\ \mathbf{y} \mid \mathbf{x} &\sim p_{\theta}(\mathbf{y} \mid \mathbf{x}) = p_{\theta}(\mathbf{y} \mid \boldsymbol{\eta}), && \text{measurement model} \end{aligned}$$

where $\boldsymbol{\eta} = \text{NeuralNet}(\mathbf{x})$.

Unfortunately, the log-likelihood for this model is intractable, so we can't directly use it as our loss function.

Instead, we're going to use an approach called *variational inference* to write a lower bound on the log-likelihood, which we'll use as our loss instead.

Variational inference introduces an approximation to the exact latent variable posterior distribution that we can sample from to impute factor scores during fitting. It's typically an isotropic Gaussian:

$$q_{\phi(\mathbf{y})}(\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}(\mathbf{y}), \boldsymbol{\sigma}^2(\mathbf{y})\mathbf{I}).$$

Amortized vs. Non-Amortized Variational Inference

In regular variational inference, we have a separate *variatonal parameters* $\phi(\mathbf{y}) = (\boldsymbol{\mu}(\mathbf{y})^\top, \boldsymbol{\sigma}(\mathbf{y})^\top)^\top$ for each person.

This becomes very computationally intensive for large sample sizes.

In amortized variational inferece, we introduce a neural net *inference model* to map from the item responses to the variational parameter values:

$$\boldsymbol{\mu}, \log \boldsymbol{\sigma} = \text{NeuralNet}_\phi(\mathbf{y}).$$

Now we share the neural net parameters ϕ across all people, making inference computationally efficient.

Learning Outcomes Checkpoint

- We've discussed the intuition behind variational autoencoders and the difference between amortized and non-amortized variational inference.
- Now we'll get into fitting VAEs.

The Variational Lower Bound

We can derive a lower bound on the observed variable likelihood as follows:

$$
\begin{aligned}
\log p(\mathbf{y}) &= \log \int p(\mathbf{y} \mid \mathbf{x}) p(\mathbf{x}) d\mathbf{x} \\
&= \log \int q(\mathbf{x} \mid \mathbf{y}) \frac{p(\mathbf{y} \mid \mathbf{x}) p(\mathbf{x})}{q(\mathbf{x} \mid \mathbf{y})} d\mathbf{x} && \text{Multiply by a constant} \\
&\geq \int q(\mathbf{x} \mid \mathbf{y}) \log \frac{p(\mathbf{y} \mid \mathbf{x}) p(\mathbf{x})}{q(\mathbf{x} \mid \mathbf{y})} d\mathbf{x} && \text{Jensen's inequality} \\
&= \int q(\mathbf{x} \mid \mathbf{y}) \log p(\mathbf{y} \mid \mathbf{x}) d\mathbf{x} - \int q(\mathbf{x} \mid \mathbf{y}) \log \frac{q(\mathbf{x} \mid \mathbf{y})}{p(\mathbf{x})} d\mathbf{x} && \text{Logarithms} \\
&= \underbrace{\mathbb{E}_{\mathbf{x} \sim q(\mathbf{x} \mid \mathbf{y})} \left[p(\mathbf{y} \mid \mathbf{x}) \right]}_{\text{expected conditional likelihood}} - \underbrace{D_{\mathrm{KL}} \left[q(\mathbf{x} \mid \mathbf{y}) \| p(\mathbf{x}) \right]}_{\text{Kullback-Leibler divergence}} .
\end{aligned}
$$

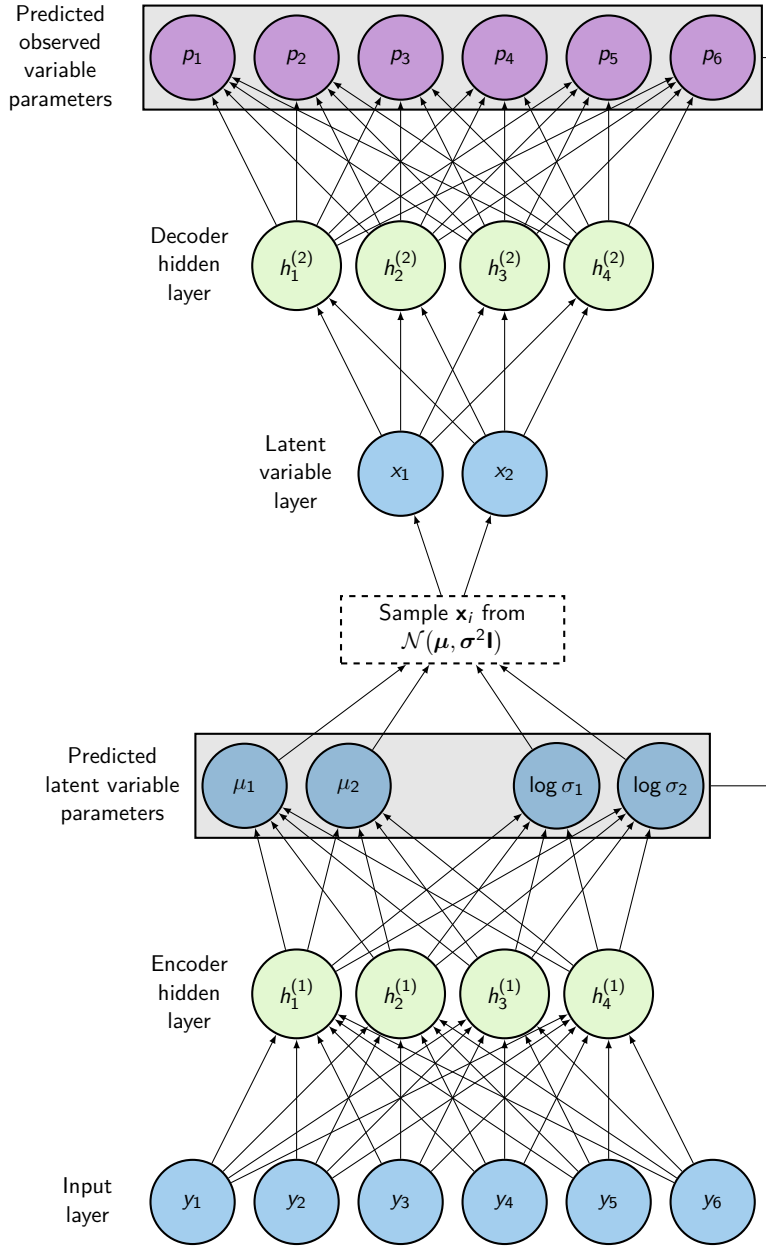
where the last line is called the *evidence lower bound* (ELBO).

ELBO is our loss function. We can show that:

$$\log p(\mathbf{y}) - \text{ELBO} = D_{\mathrm{KL}}[q(\mathbf{x} \mid \mathbf{y}) \mid p(\mathbf{x} \mid \mathbf{y})],$$

i.e., the tightness of ELBO depends on the accuracy of our posterior approximation.

What's Actually Happening in VAEs



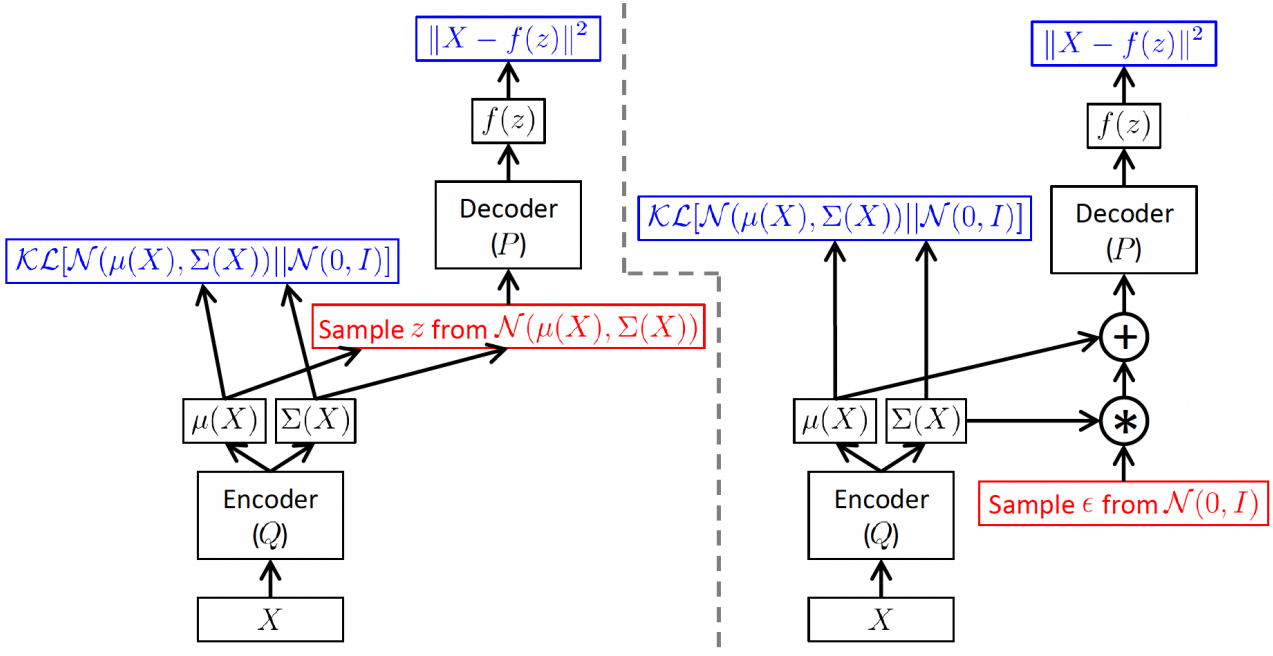
$$\underbrace{\mathbb{E}_{\mathbf{x}\sim\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\sigma}^2\mathbf{I})}\left[\prod_{j=1}^{6}\text{Bernoulli}(p_j)\right]}_{\text{expected conditional likelihood}}$$

$$\underbrace{D_{\text{KL}}\left[\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\sigma}^2\mathbf{I})\|\mathcal{N}(\mathbf{0},\mathbf{I})\right]}_{\text{Kullback-Leibler divergence}}$$

Reparameterization Trick

Derivatives don't pass through the sampling operation, so we use a *reparameterization trick* to make sampling differentiable:

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$
$$\mathbf{x} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}.$$



This produces a low variance gradient estimator for the neural net parameters ϕ, but is tricky to implement with discrete latent variables.

Tightening the Lower Bound via Importance Sampling

We can use importance sampling to tighten the ELBO:

$$\begin{aligned} \log p_{\boldsymbol{\delta}}(\mathbf{y}) &\geq \mathbb{E}_{\mathbf{x}_{1:R}}\left[\log \frac{1}{R}\sum_{r=1}^{R} w_r\right] &&= \text{IW-ELBO} \\ &\geq \mathbb{E}_{\mathbf{x}}[\log w] &&= \text{ELBO}, \end{aligned}$$

where $w_r = p_{\boldsymbol{\delta}}(\mathbf{x}_r, \mathbf{y})/q_{\boldsymbol{\phi}}(\mathbf{x}_r \mid \mathbf{y})$.

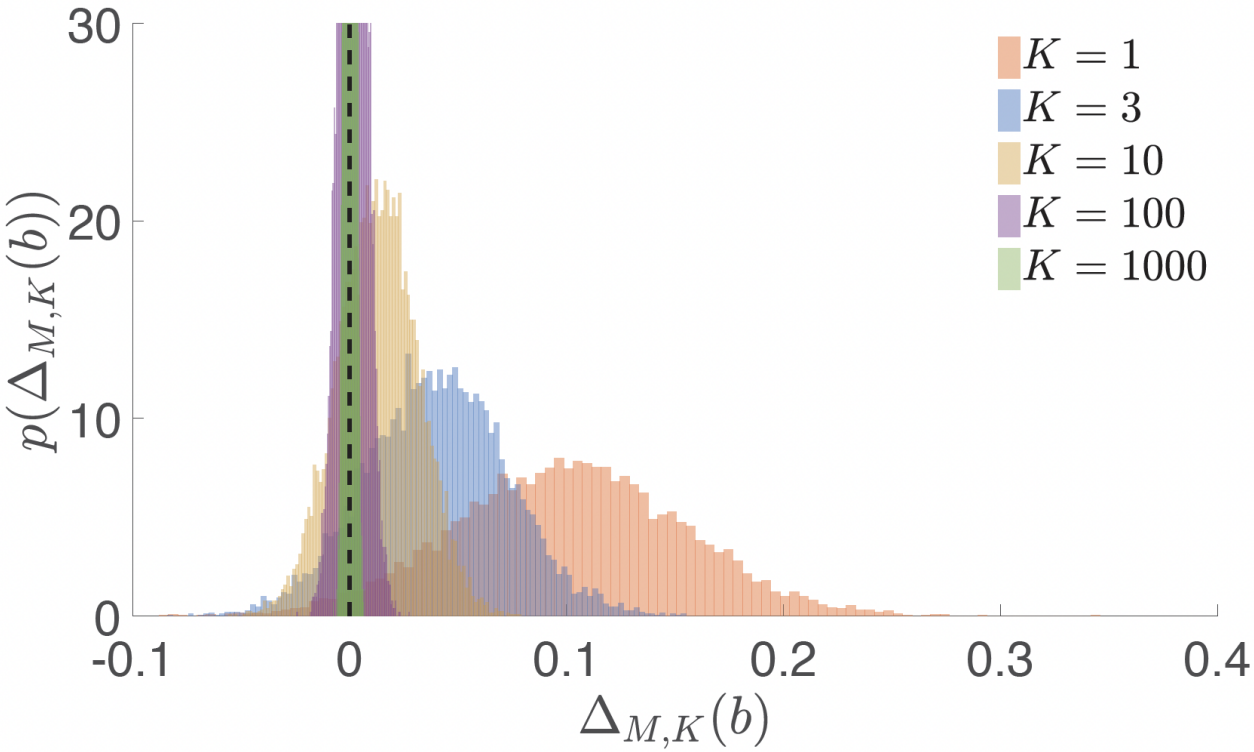
This model is called an importance-weighted autoencoder (IWAE).

As $R \to \infty$:

- The IW-ELBO converges monotonically to the true log-likelihood.
- The approximate posterior converges to the exact posterior.

Burda, Grosse, and Salakhutdinov, (2016). Importance weighted autoencoders.

Signal-to-Noise Vanishing



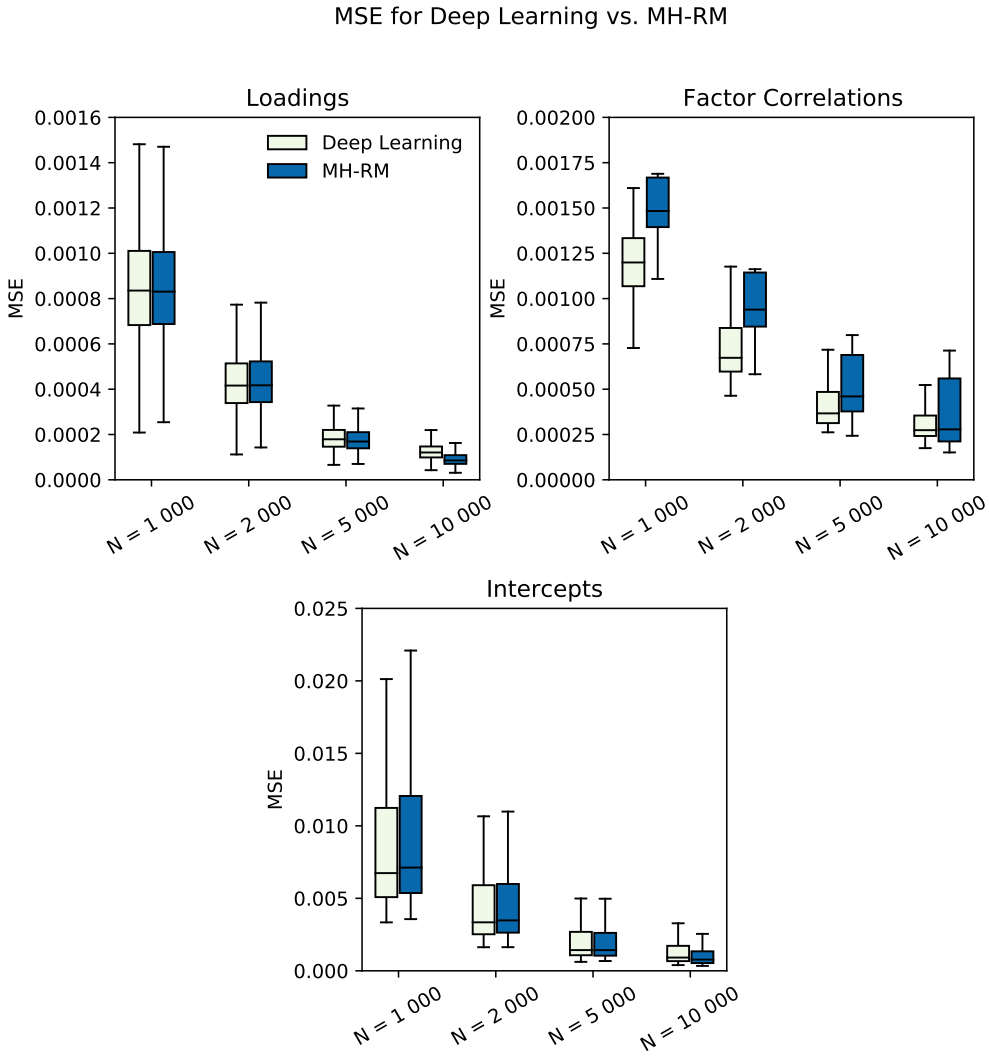
(a) IWAE inference network gradient estimates

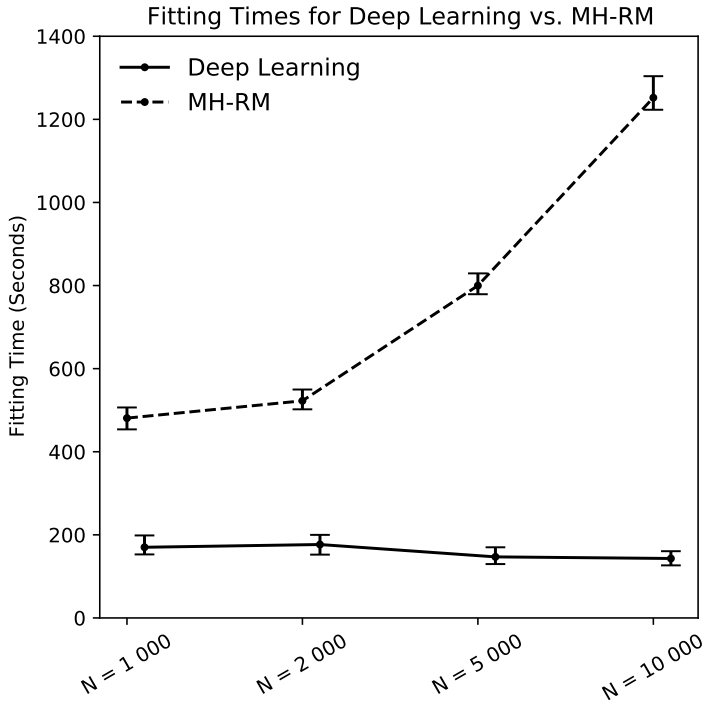
Can be fixed with a "minor" modification to the inference network gradient. See my paper for details.

Rainforth et al. (2018). Tighter variational bounds are not necessarily better.

How Does This Work for Fitting IRT Models?

Urban, C. J. & Bauer, D. J. (2021). A deep learning algorithm for high-dimensional exploratory item factor analysis. *Psychometrika*. *86* (1), 1-29.





Tons of Extensions!

- Longitudinal data — we'd used an RNN inference network
- Pretty much any measurement model and/or latent prior
- Layers of latent variables
- Multilevel / SEM-type models

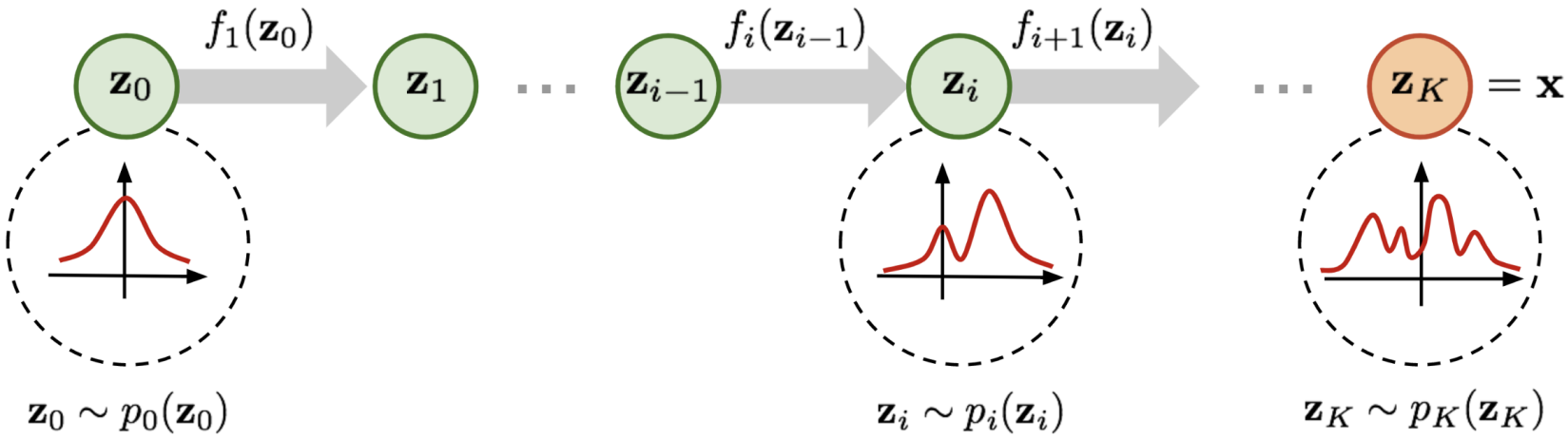
Normalizing Flows for Latent Density Estimation

We apply a sequence of transformations to some simple base distribution to transform it into a more complicated distribution.

If the transformations are diffeomorphisms, we can evaluate the final density using the change of variables formula:

$$p_Y(\mathbf{y}) = \left| \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right|^{-1} p_X(\mathbf{x}).$$

This can be applied to the latent variable prior!



Software

- Tensorsem — fitting SEMs using backprop and stochastic gradient methods
- DeepIRTools — fiitting latent factor models using importance-weighted amortized variational inference
- Pyro and NumPyro — build general probabilistic models in PyTorch and Jax
- Tensorflow Probability — work with probabilistic models in Tensorflow

Final Checkpoint

- We've finished discussing fitting VAEs and some extensions.
- On to programming!