

Lecture 25

# Generative Models and Variational Inference

# Tutorial paper due

Tues May 1st, 11.59pm

Exam, 2-3 questions, 1-2 simple, 1 a bit hard (bit more work)

Exam released same night. You will have 10 days.

- This course has mostly been about Unsupervised Learning
- That is, estimating a  $p(x)$  from data
- Supervised learning can be cast into this density estimation paradigm:  $p(x, y)$
- these are "predictive" distributions
- We use some latent variables  $z$ , which may be clusters, or estimation parameters  $\theta$

# Latent Variables

- key concept: full data likelihood vs partial data likelihood
- probabilistic model is a *joint distribution*  $p(\mathbf{x}, \mathbf{z})$ , the full data likelihood
- with observed variables  $\mathbf{x}$  corresponding to data, and latent variables  $\mathbf{z}$

# Concrete Formulation of unsupervised learning

Estimate Parameters by  $\mathbf{x}$ -MLE:

$$\begin{aligned}l(\mathbf{x}|\lambda, \mu, \Sigma) &= \sum_{i=1}^m \log p(x_i|\lambda, \mu, \Sigma) \\ &= \sum_{i=1}^m \log \sum_z p(x_i|z_i, \mu, \Sigma) p(z_i|\lambda)\end{aligned}$$

Not Solvable analytically! EM and Variational. Or do MCMC.

ENM

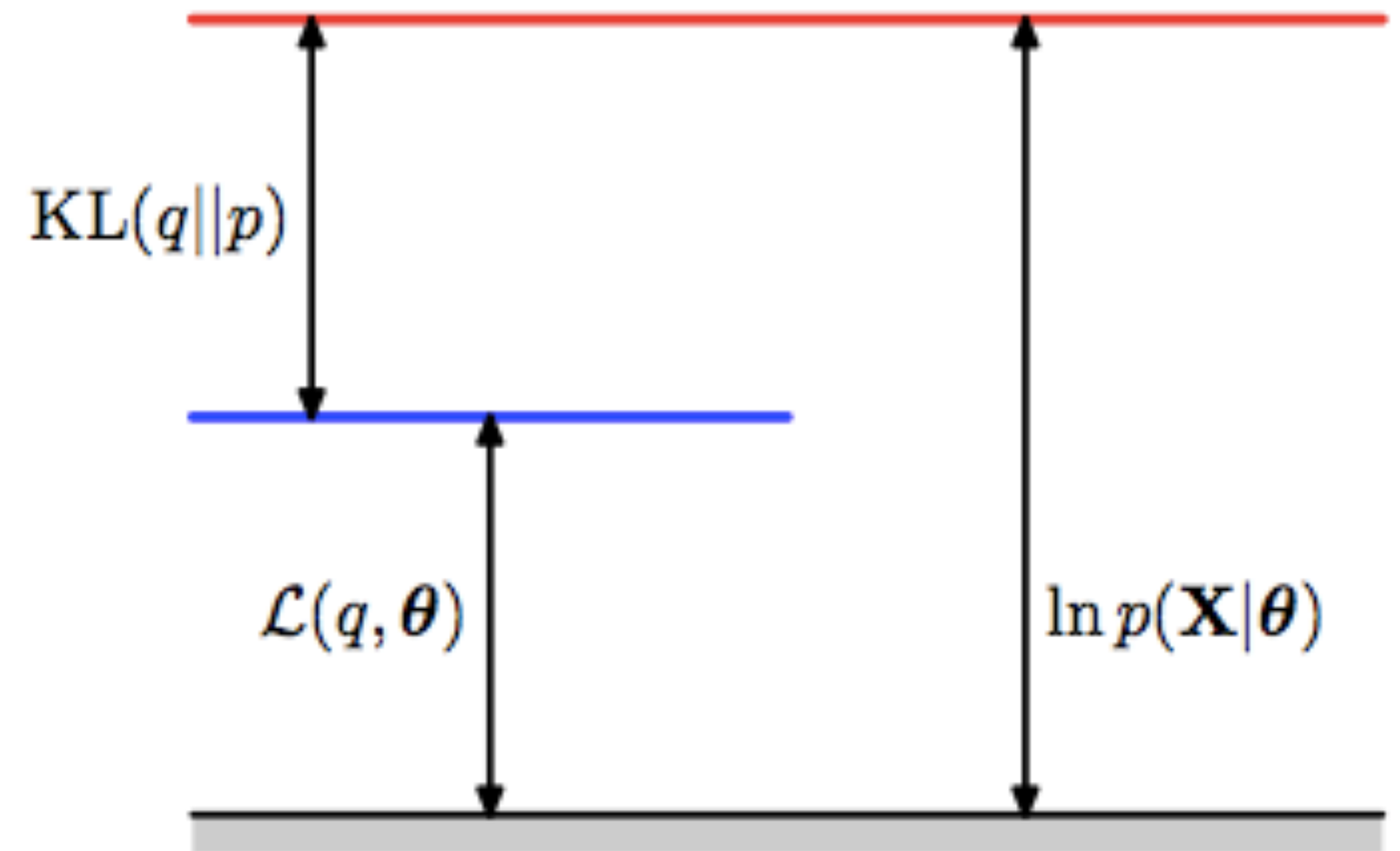
x-data likelihood: carry along  $\theta$

$$\log p(x|\theta) = E_q\left[\log \frac{p(x, z|\theta)}{q}\right] + D_{KL}(q, p)$$

If we define the ELBO or Evidence Lower bound as:

$$\mathcal{L}(q, \theta) = E_q\left[\log \frac{p(x, z|\theta)}{q}\right]$$

then  $\log p(x|\theta) = \text{ELBO} + \text{KL-divergence}$



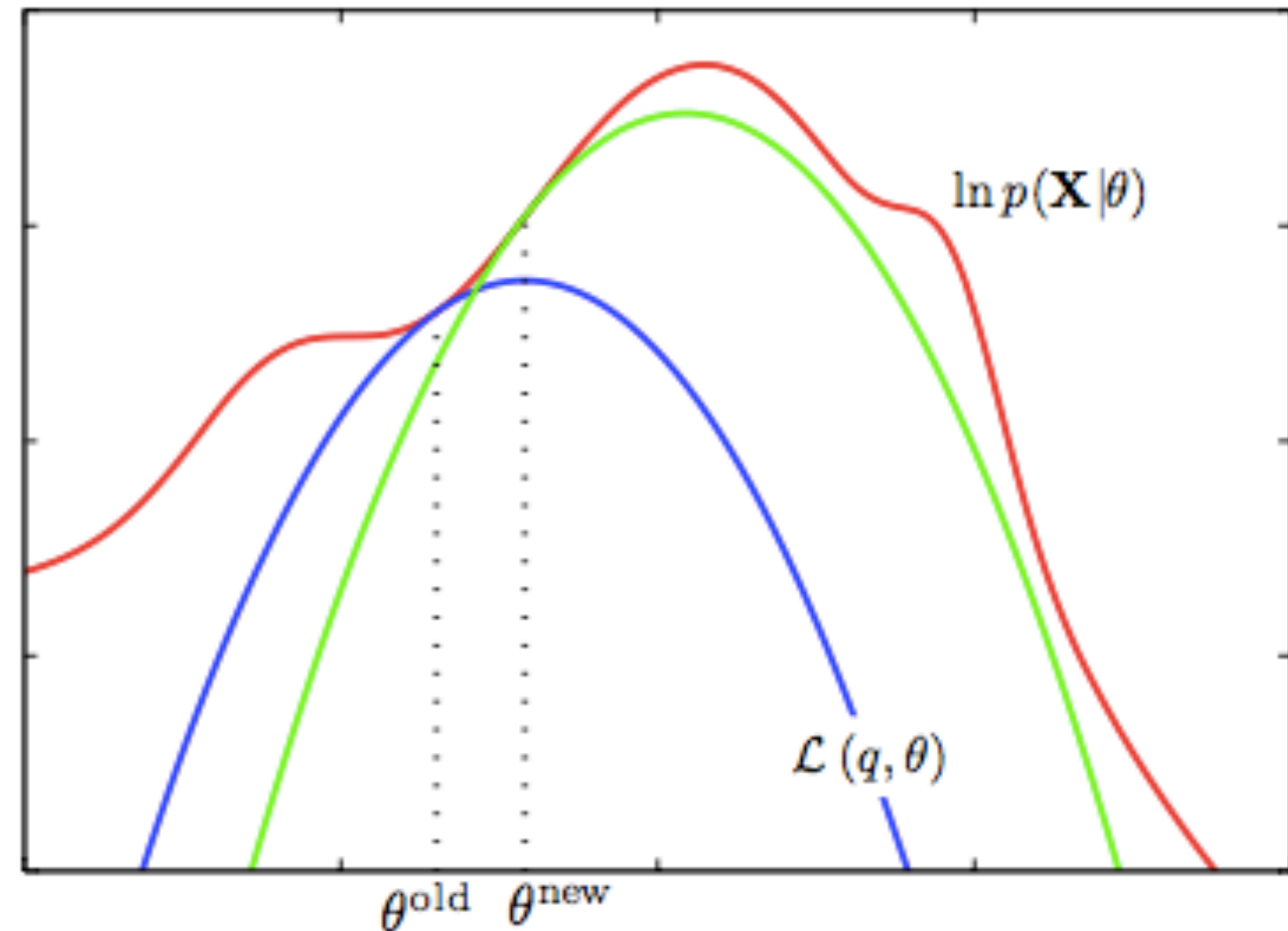
- KL divergence only 0 when  $p = q$  exactly everywhere
- minimizing KL means maximizing ELBO
- ELBO  $\mathcal{L}(q, \theta)$  is a lower bound on the log-likelihood.
- ELBO is average full-data likelihood minus entropy of  $q$ :

$$\mathcal{L}(q, \theta) = E_q \left[ \log \frac{p(x, z | \theta)}{q} \right] = E_q [\log p(x, z | \theta)] - E_q [\log q]$$



# Process

1. Start with  $p(x|\theta)$  (red curve),  $\theta_{old}$ .
2. Until convergence:
  1. E-step: Evaluate  $q(z, \theta_{old}) = p(z|x, \theta_{old})$  which gives rise to  $Q(\theta, \theta_{old})$  or  $ELBO(\theta, \theta_{old})$  (blue curve) whose value equals the value of  $p(x|\theta)$  at  $\theta_{old}$ .
  2. M-step: maximize  $Q$  or  $ELBO$  wrt  $\theta$  to get  $\theta_{new}$ .
3. Set  $\theta_{old} = \theta_{new}$



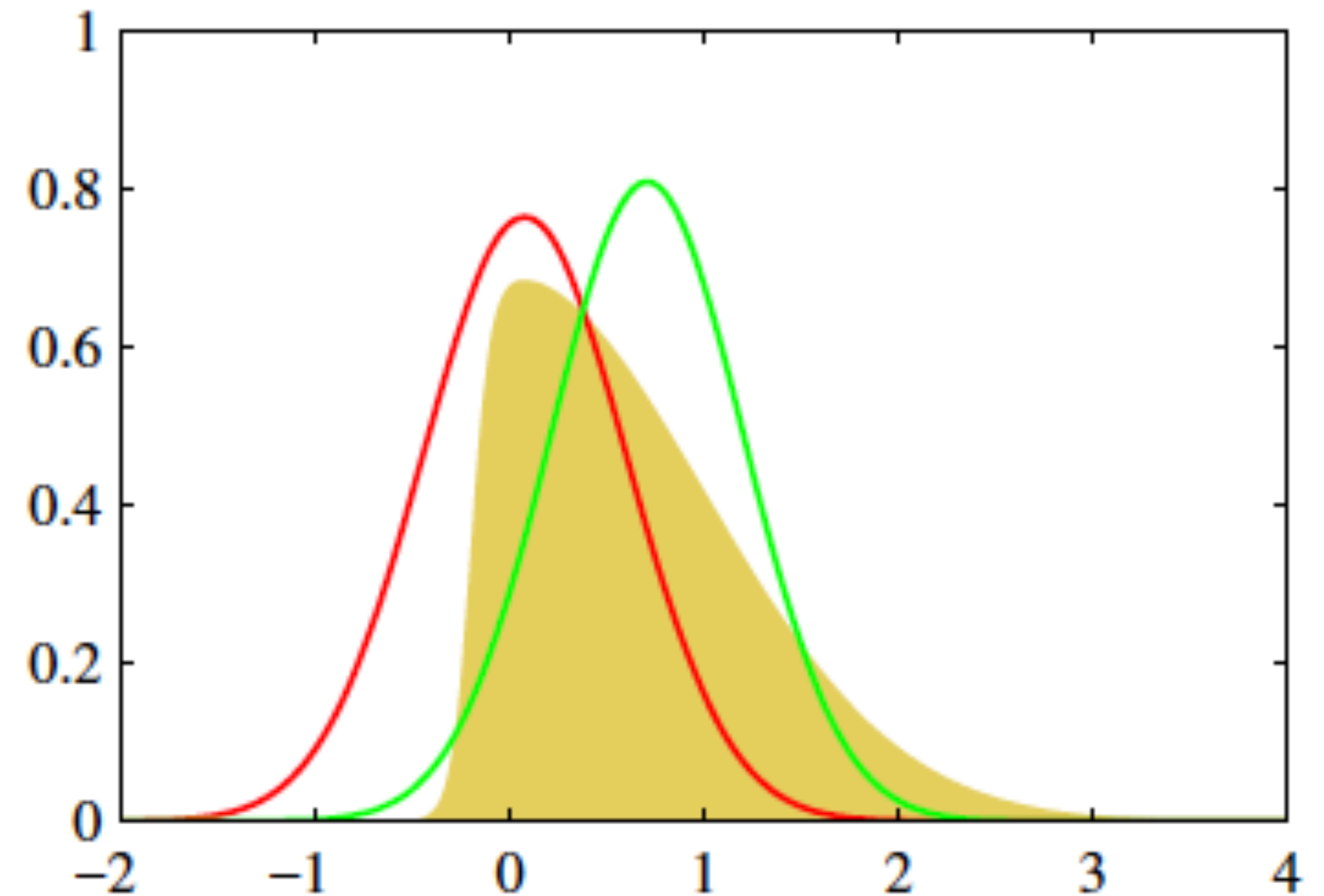
# VARIATIONAL INFERENCE

# Core Idea

$z$  is now all parameters. Don't distinguish from  $\theta$ .

Restricting to a family of approximate distributions  $D$  over  $z$ , find a member of that family that minimizes the KL divergence to the exact posterior. An optimization problem:

$$q^*(z) = \arg \min_{q(z) \in D} KL(q(z) || p(z|x))$$



# VI vs MCMC

## MCMC

---

More computationally intensive

---

Guarantees producing asymptotically exact samples from target distribution

---

Slower

---

Best for precise inference

## VI

---

Less intensive

---

No such guarantees

---

Faster, especially for large data sets and complex distributions

---

Useful to explore many scenarios quickly or large data sets

# Basic Setup in EM

Recall that  $KL + ELBO = \log(p(x))$ ,  
 $ELBO(q) = E_q[\log(p(z, x))] - E_q[\log(q(z))]$

EM alternates between computing the expected complete log likelihood according to  $p(z|x)$  (the E step) and optimizing it with respect to the model parameters (the M step).

EM assumes the expectation under  $p(z|x)$  is computable and uses it in otherwise difficult parameter estimation problems.

## Basic Setup in VI

$KL + ELBO = \log(p(x))$ : ELBO bounds log(evidence)

$$ELBO(q) = E_q \left[ \log \frac{p(z, x)}{q(z)} \right] = E_q \left[ \log \frac{p(x|z)p(z)}{q(z)} \right] = E_q [\log p(x|z)] + E_q \left[ \log \frac{p(z)}{q(z)} \right]$$

$$\implies ELBO(q) = E_{q(z|x)} [\log(p(x|z))] - KL(q(z|x) || p(z))$$

(likelihood-prior balance)

Mean Field: Find a  $q$  such that:

$KL + ELBO = \log(p(x))$ : KL minimized means ELBO maximized.

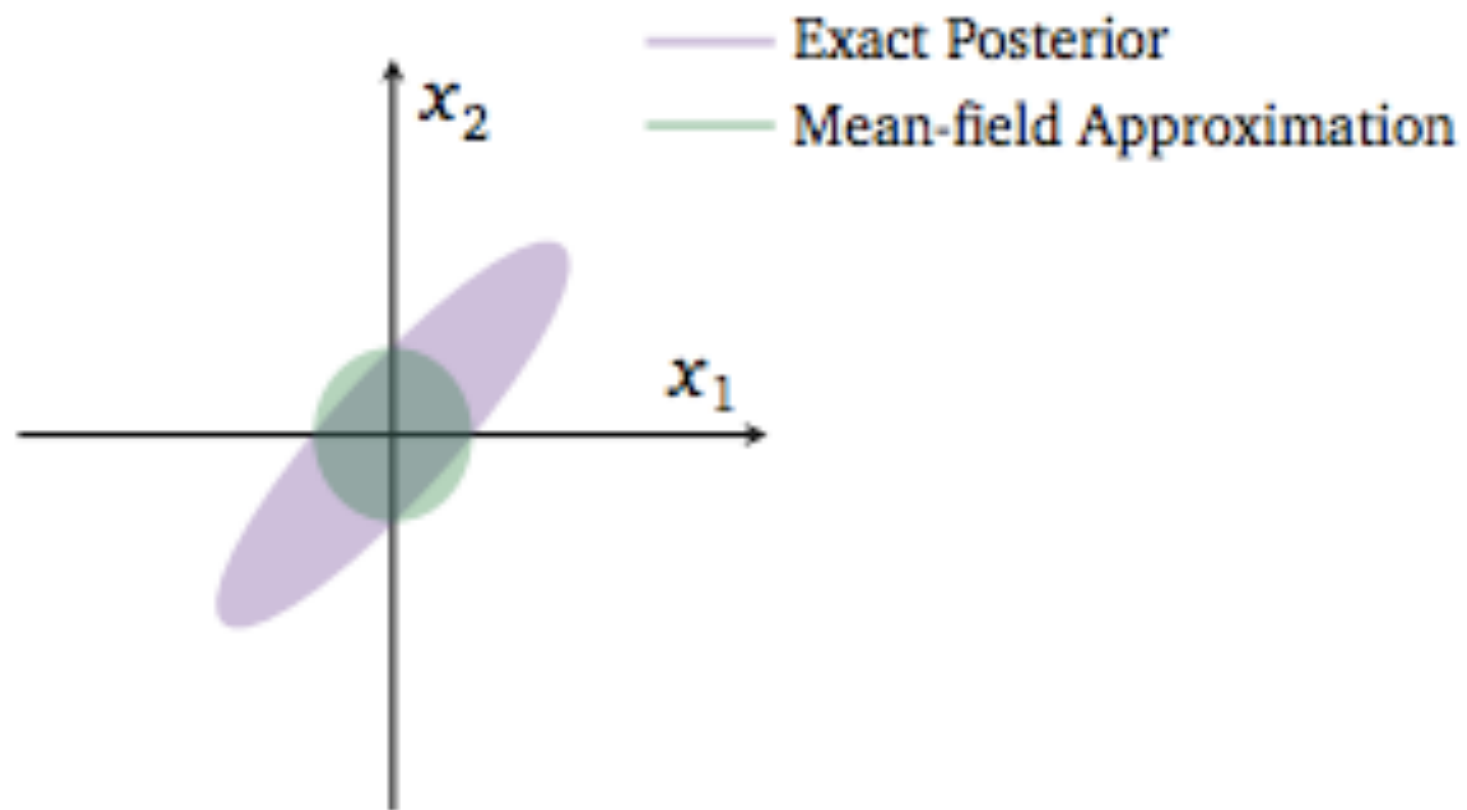
Choose a "mean-field"  $q$  such that:

$$q(z) = \prod_{j=1}^m q_j(z_j)$$

Each individual latent factor can take on any paramteric form corresponding to the latent variable.

## Example

$$q(z) = \prod_{j=1}^m q_j(z_j)$$



a 2D Gaussian Posterior is approximated by a mean-field variational structure with independent gaussians in the 2 dimensions

The variational posterior in green cannot capture the strong correlation in the original posterior because of the mean field approximation.



# Optimization: CAVI

*Coordinate ascent mean-field variational inference*

maximizes ELBO by iteratively optimizing each variational factor of the mean-field variational distribution, while holding the others fixed.

Define Complete Conditional of  $z_j = p(z_j | \mathbf{z}_{-j}, \mathbf{x})$

# Algorithm

**Input:**  $p(x, z)$  with data set  $x$ , **Output:**  $q(z) = \prod_j q_j(z_j)$

**Initialize:**  $q_j(z_j)$

while ELBO has not converged (or  $z$  have not converged):  
  for each  $j$ :

$$q_j \propto \exp(E_{-j}[\log p(z_j | z_{-j}, x)])$$

  compute ELBO

where the expectations above are with respect to the variational distribution over  $\mathbf{z}_{-j}$ :

$$\prod_{l \neq j} q_l(z_l)$$

**Assertion:**  $q_j^*(z_j) \propto \exp\{E_{-j}[\log(p(z_j | \mathbf{z}_{-j}, \mathbf{x}))]\}$   
 $\implies q_j^*(z_j) \propto \exp\{E_{-j}[\log(p(z_j, \mathbf{z}_{-j}, \mathbf{x}))]\}$

(because the mean-field family assumes that all the latent variables are independent)

## Example: "Fake :-) Gaussian"

```
data = np.random.randn(100)
with pm.Model() as model:
    mu = pm.Normal('mu', mu=0, sd=1)
    sd = pm.HalfNormal('sd', sd=1)
    n = pm.Normal('n', mu=mu, sd=sd, observed=data)
```

Assume Gaussian posteriors for  $\mu$  and  $\log(\sigma)$ . So, for e.g.,

$$\mu \sim N(\mu_\mu, \sigma_\mu^2), \log(\sigma) \sim N(\mu_\sigma, \sigma_\sigma^2)$$

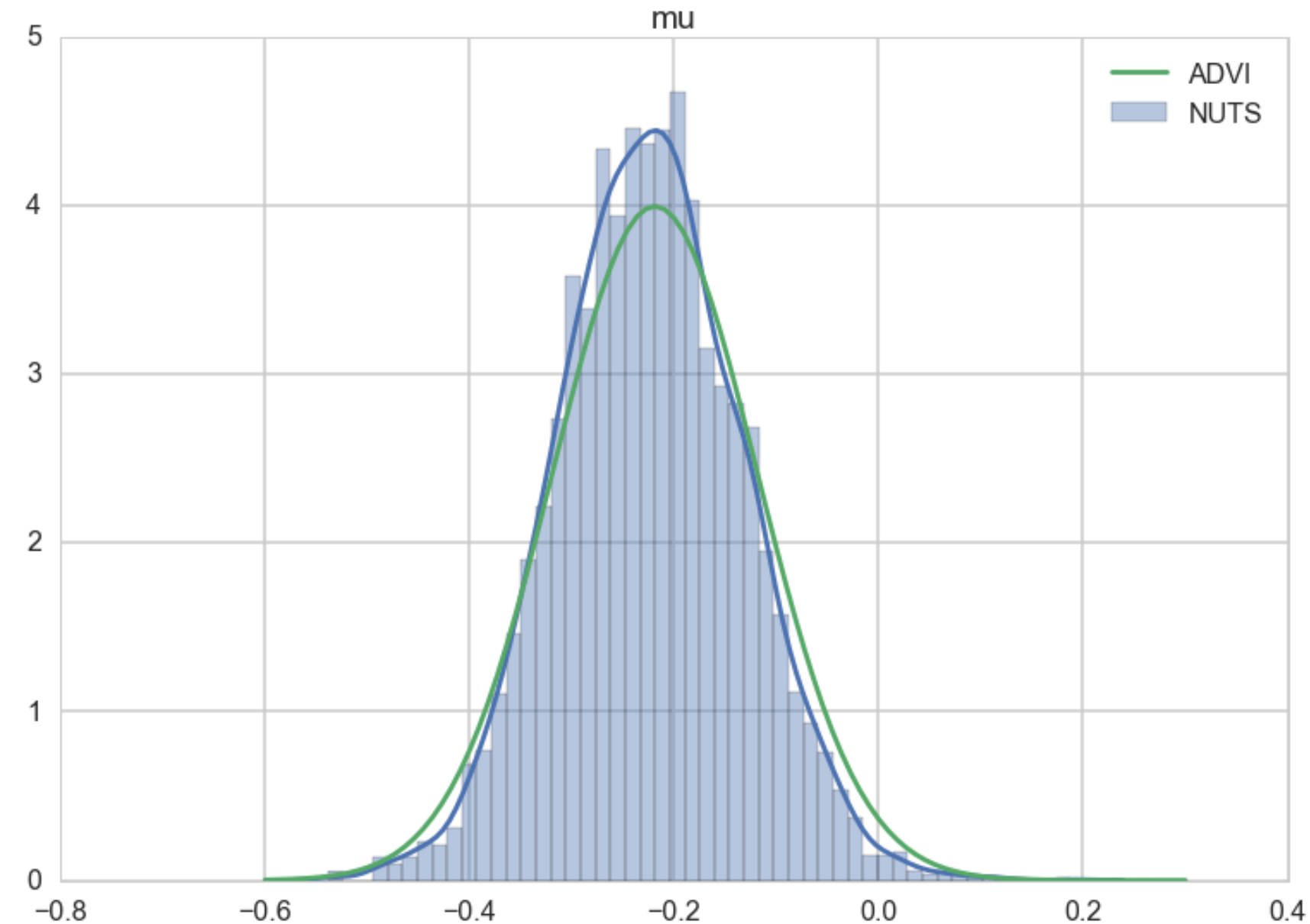
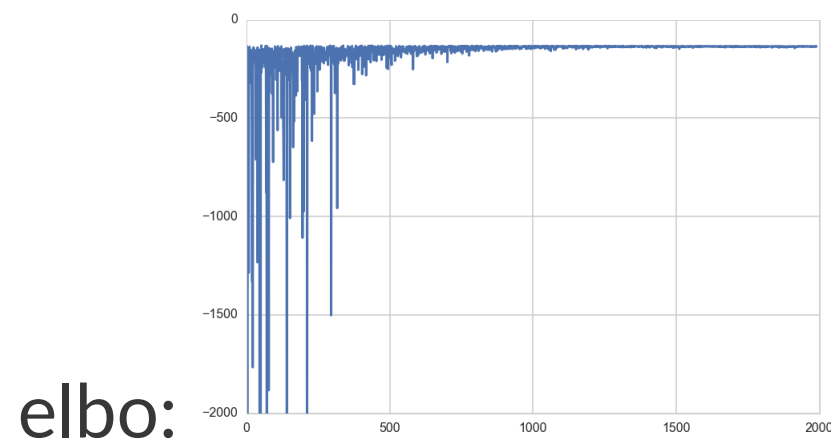
**ADNM**

## Core Idea:

- CAVI does not scale, needs graduate student descent
- Use gradient descent instead
- Use minibatches to do it on less data
- do it automatically using automatic differentiation

# ADVI in pymc3

```
data = np.random.randn(100)
with pm.Model() as model:
    mu = pm.Normal('mu', mu=0, sd=1, testval=0)
    sd = pm.HalfNormal('sd', sd=1)
    n = pm.Normal('n', mu=mu, sd=sd, observed=data)
advifit = pm.ADVI(model=model)
advifit.fit(n=50000)
elbo = -advifit.hist
plt.plot(elbo[::10]);
```



# What does ADVI do?

1. Transformation of latent parameters (**T** transform)
  - reparametrize mean field parameters to the real line
2. Standardization transform for posterior to push gradient inside expectation (**S** transform)
3. Monte-Carlo estimate of expectation
4. Hill-climb using automatic differentiation



Remember:

$$ELBO(q) = E_q[\log(p(z, x))] - E_q[\log(q(z))]$$

Need

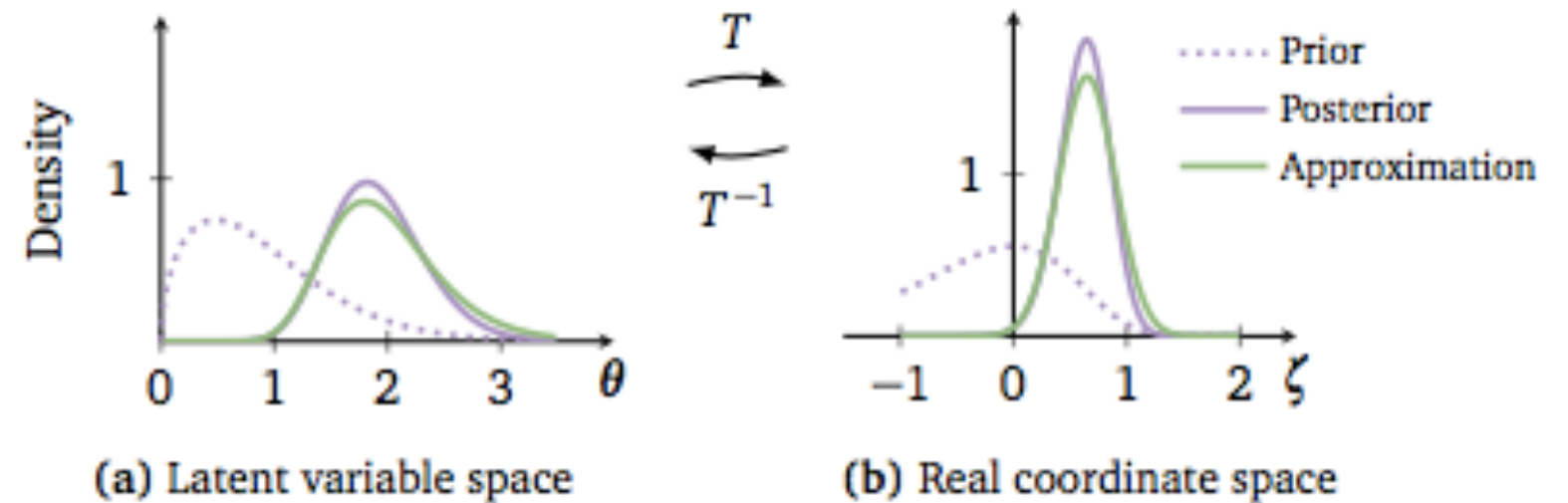
$$\nabla_{\eta} \mathcal{L} = E[\nabla_{\eta} [\log p(x, T^{-1}(S^{-1}(\eta))) + \log(\det(J_{T^{-1}}(S^{-1}(\eta)))))]$$

where  $S$  is the first transform and  $T$  is the standardization.

# (1) T-Transformation

- Latent parameters are transformed to representations where the 'new" parameters are unconstrained on the real-line. Specifically the joint  $p(x, \theta)$  transforms to  $p(x, \eta)$  where  $\eta$  is un-constrained.
- Minimize the KL-divergence between the transformed densities.
- This is done for *ALL* latent variables.
- Thus use the same variational family for ALL parameters, and indeed for ALL models,

- Discrete parameters must be marginalized out.
- Optimizing the KL-divergence implicitly assumes that the support of the approximating density lies within the support of the posterior. These transformations make sure that this is the case
- First choose as our family of approximating densities mean-field normal distributions. We'll transform the always positive  $\sigma$  params by simply taking their logs.



## (2) S-transformation

- we must maximize our suitably transformed ELBO.
- we are optimizing an expectation value with respect to the transformed approximate posterior. This posterior contains our transformed latent parameters so the gradient of this expectation is not simply defined.
- we want to push the gradient inside the expectation. For this, the distribution we use to calculate the expectation must be free of parameters

### (3) Compute the expectation

As a result of this, we can now compute the integral as a monte-carlo estimate over a standard Gaussian--superfast, and we can move the gradient inside the expectation (integral) to boot. This means that our job now becomes the calculation of the gradient of the full-data joint-distribution.

## (4) Calculate the gradients

We can replace full  $x$  data by just one point (SGD) or mini-batch (some- $x$ ) and thus use noisy gradients to optimize the variational distribution.

An adaptively tuned step-size is used to provide good convergence.

# Relaxing the mean-field approximation

- Full-Rank ADVI: model covariance
- Normalizing Flows
- Operator Variational Inference: allows generalization of many algorithms under one umbrella

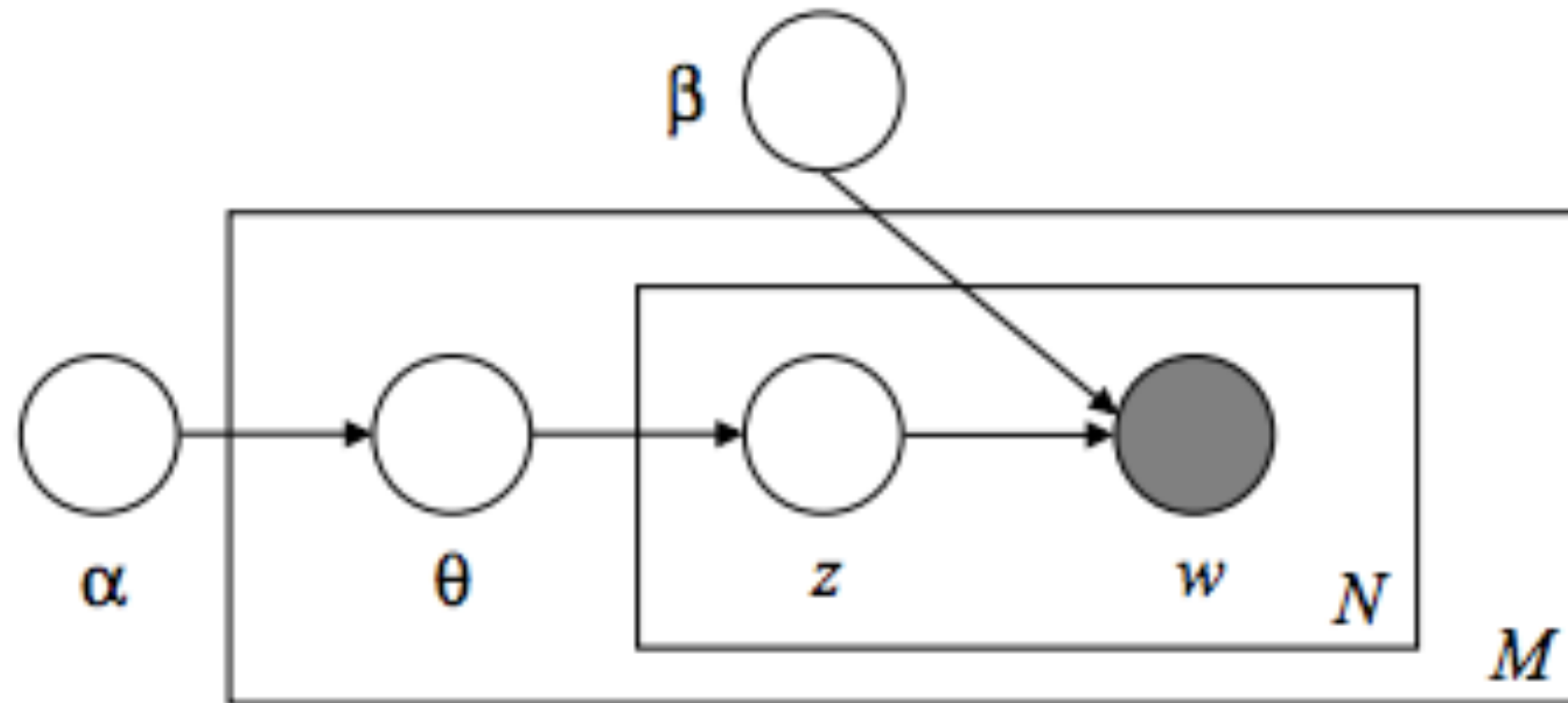
(all implemented in pymc3)

# How good is variational Bayes?

- its used heavily for models like LDA (latent-dirichlet allocation)
- but surprisingly the "goodness-of-fit" of the posterior approximation has been handled on a case by case basis
- until now: see [Yao et. al](#)



# LDA: a generative model



See [Blei et. al.](#)

LDA assumes the following generative process for *each* document  $w$  in a corpus  $D$ :

1. Choose  $N \sim \text{Poisson}(\xi)$ .
2. Choose  $\theta \sim \text{Dir}(\alpha)$ .
3. For each of the  $N$  words  $w_n$  (from vocab size  $V$ ):
  1. Choose a topic  $z_n \sim \text{Multinomial}(\theta)$  (size  $k$ ).
  2. Choose a word  $w_n$  from  $p(w_n | z_n, \beta)$ ,  $\beta$  size  $V \times k$ , a multinomial probability conditioned on the topic  $z_n$ .

## Two ideas from Yao et. al.

- pareto shape parameter  $k$  from PSIS tells you goodness of fit (see [here](#) for @junpenglao pymc3 implementation, WIP). The idea comes from the process of smoothing in LOOCV estimation
- VSBC (variational simulation based calibration) : Extends calibration from Bayesian Workflow to variational case. pymc3 experimentation by @junpenglao [here](#), WIP

# Pareto Smoothed Importance Sampling

Want  $E_p[h(\theta)]$ . But we calculate  $E_q[h(\theta)] = (1/S) \sum_s h(\theta_s)$  which is biased.

Use importance sampling:  $E_p[h(\theta)] = \frac{\sum_s w_s h(\theta_s)}{\sum_s w_s}$  where  $w_s = p(\theta_s, y)/q$ .  $w_s$

may have large or infinite variance.

Use PSIS: fit shape  $k$  Pareto to  $M$  largest  $w_s$  and replace them by expected values of corresponding order statistics under the pareto. Also truncate all weights at raw maximum  $w_s$ . Use joint as pareto cares not about multiplying factors.

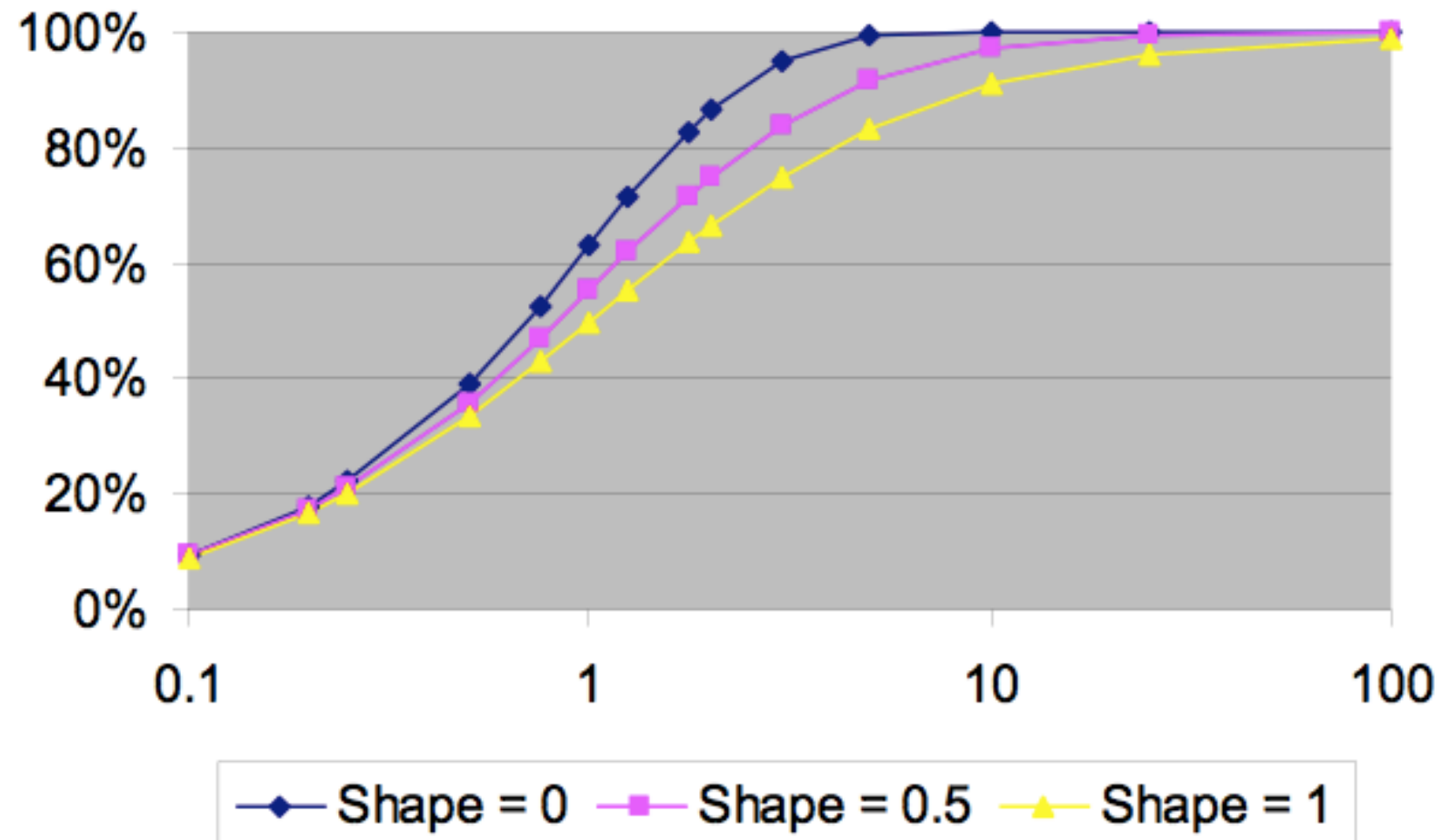
M empirically set as  $\min(S/5, 3\sqrt{S})$ .

Result from extreme value theory (Pickands–Balkema–de Haan theorem): conditional excess distribution function is a generalized pareto

$$p(y|\mu, \sigma, k) = \begin{cases} \frac{1}{\sigma} \left( 1 + k \left( \frac{y - \mu}{\sigma} \right) \right)^{-\frac{1}{k}-1}, & k \neq 0. \\ \frac{1}{\sigma} \exp \left( -\frac{y - \mu}{\sigma} \right), & k = 0. \end{cases}$$

$k < 0.5$  great, ok between 0.5 and 0.7, not so good after 0.7, weights too large.

[source](#)

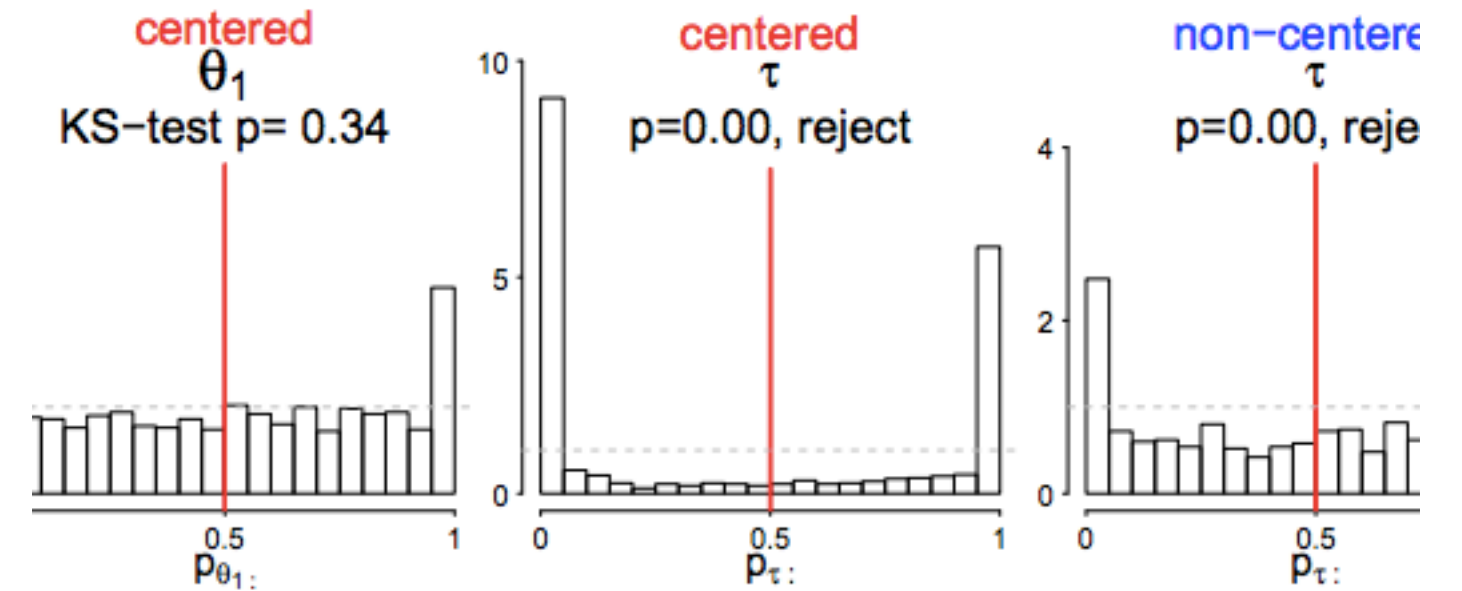
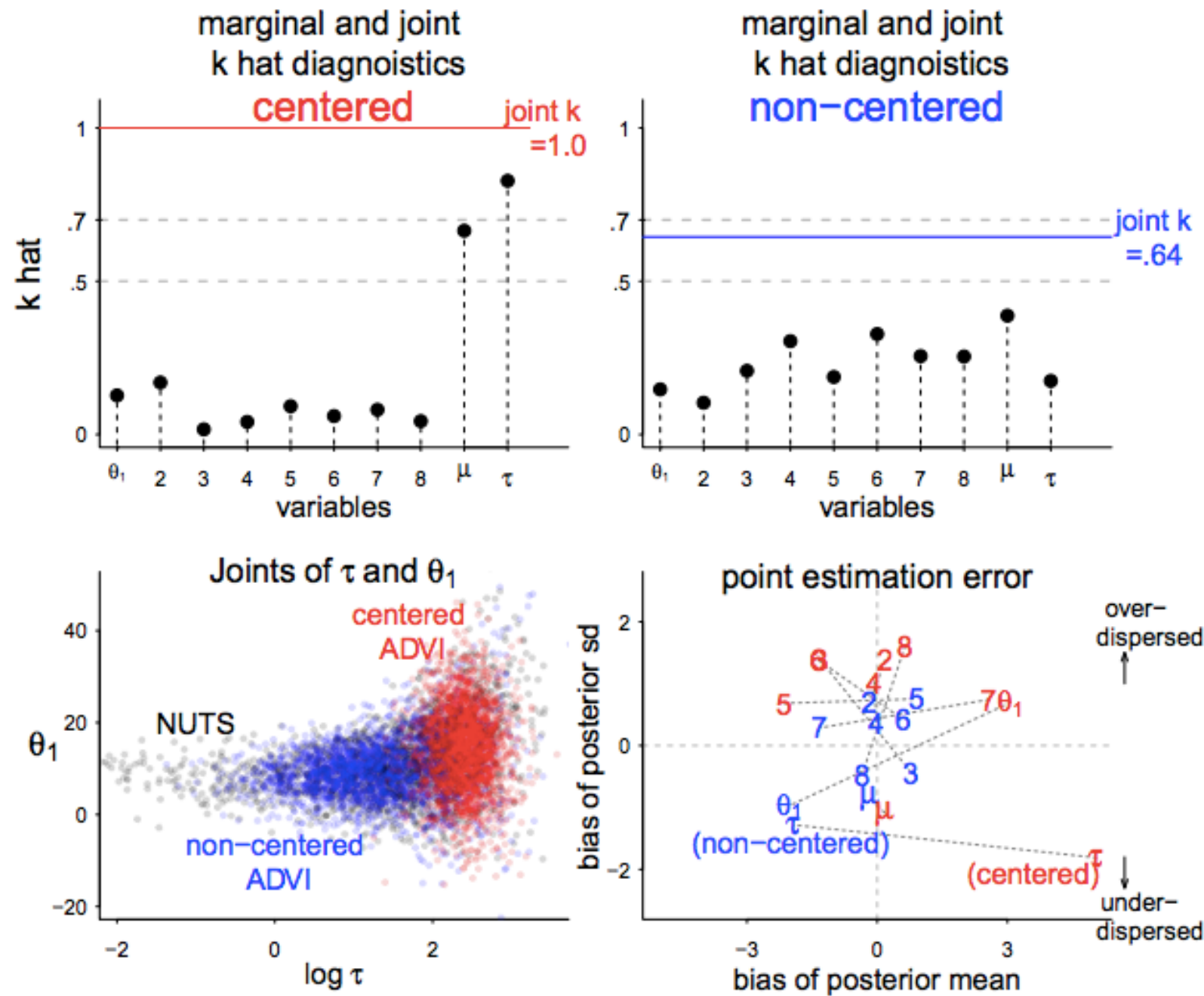


# VSBC

- basic idea from bayesian workflow, posterior from data simulated from prior ( $\theta_0 \sim p(\theta)$ ) should look like the prior. That is, ideally order statistics uniform
- in VSBC fit the posterior variationally. Will have some mismatch
- quantify mismatch by asymmetry in histogram of ith marginal calibration probabilities  $p_{ij} = P_q(\theta_i < [\theta_j^0]_i)$

Left: ADVI posterior and pareto shape statistics

Below: VSBC histogram



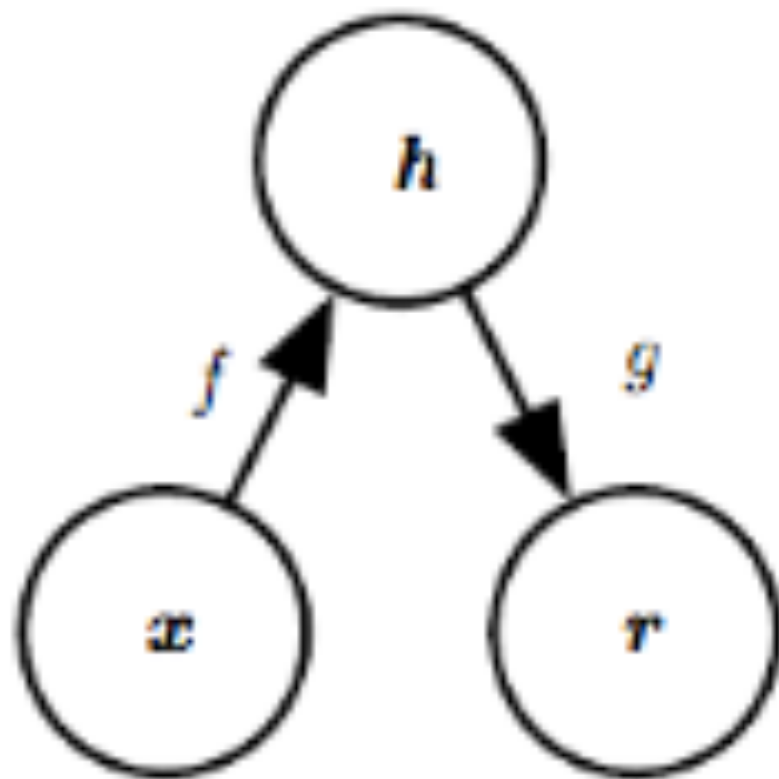
# Why use VB

- simply not possible to do inference in large models
- inference in neural networks: understanding robustness, etc
- hierarchical neural networks (perhaps on exam)
- Mixture density networks: mixture parameters are fitted using ANNs
- extension to **generative semisupervised learning**
- **variational autoencoders**



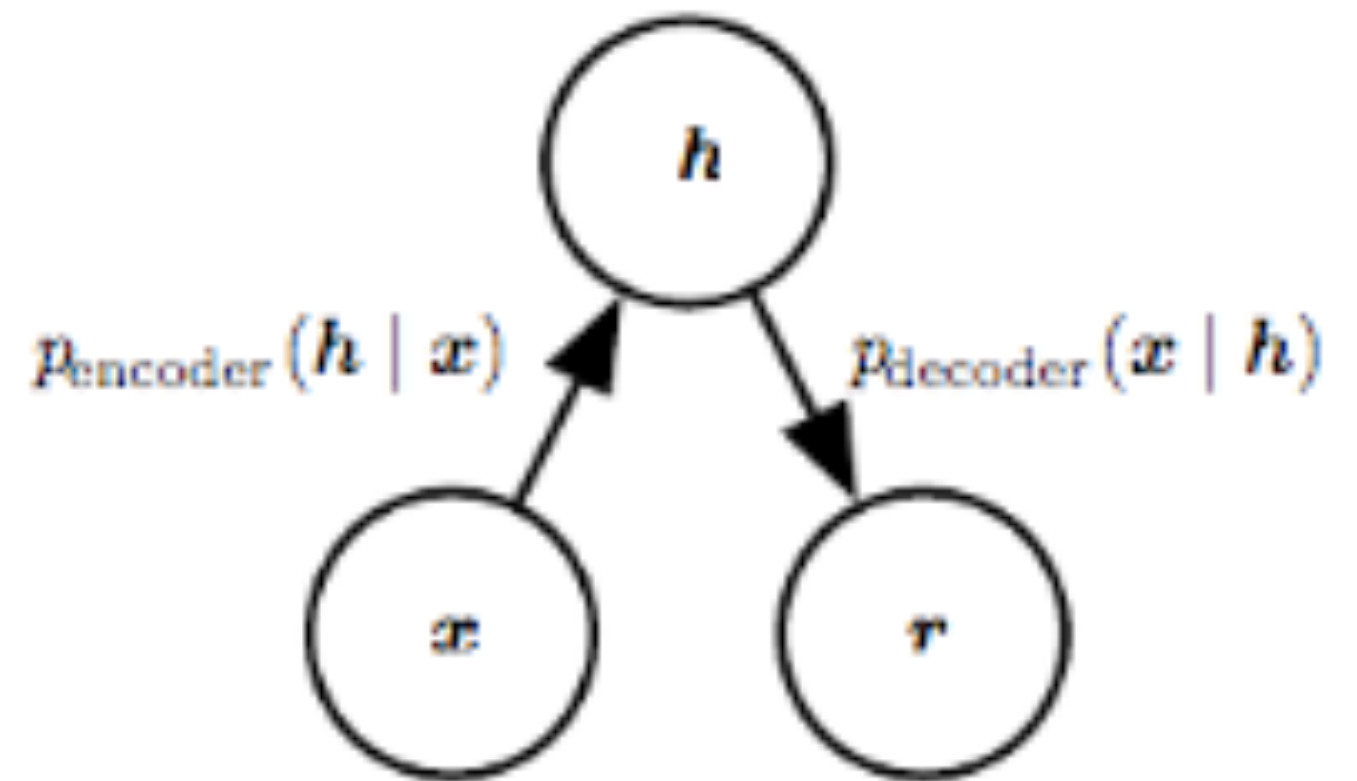
# Variational Autoencoders

# Autoencoders: basic idea



- $h$  is the representation. An *undercomplete* autoencoder makes  $h$  of smaller dimension than  $x$
- $f$  is the encoder and  $g$  the decoder
- simplest idea: minimize  $L(x, g(f(x)))$
- can regularize instead of being undercomplete

- can think of an autoencoder as a way of approximately training a generative model.
- the features of the autoencoder describe the latent variables that explain the input
- can go deep!
- generalize to a stochastic autoencoder. The standard autoencoder then is a specific hidden state  $h$  or  $z$



# Variational Autoencoder

- just as in ADVI, we want to learn an approximate "encoding posterior"  $p(z|x)$
- note that we have now again gone back to thinking of  $z$  as a (possibly) deep latent variable, or "representation".

We know how to do this:

## ELBO maximization

## Basic Setup in VI

$KL + ELBO = \log(p(x))$ : ELBO bounds log(evidence)

$$ELBO(q) = E_q \left[ \log \frac{p(z, x)}{q(z)} \right] = E_q \left[ \log \frac{p(x|z)p(z)}{q(z)} \right] = E_q [\log p(x|z)] + E_q \left[ \log \frac{p(z)}{q(z)} \right]$$

$$\implies ELBO(q) = E_{q(z|x)} [\log(p(x|z))] - KL(q(z|x) || p(z))$$

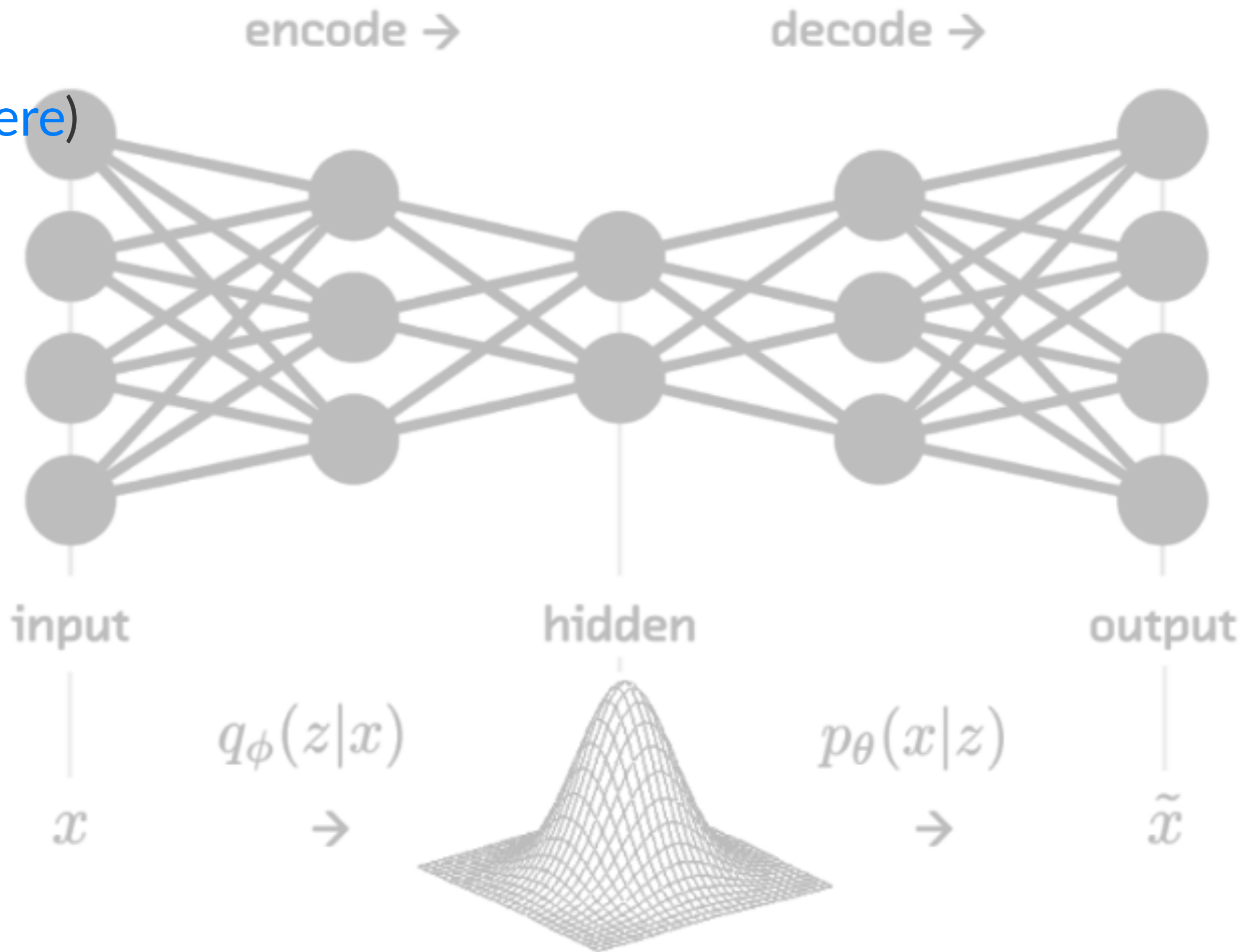
(likelihood-prior balance)

# The Game

$$ELBO(q) = E_{q(z|x)} [\log(p(x|z))] - KL(q(z|x) || p(z))$$

- get  $z$  samples coming from  $x$ ,  $q(z|x)$ - to be close to some prior,  $p(z)$ , typically chosen as an isotropic gaussian...the regularization term
- first term is called "reconstruction loss", or "capacity of model to generate something like the data".

(from [here](#))



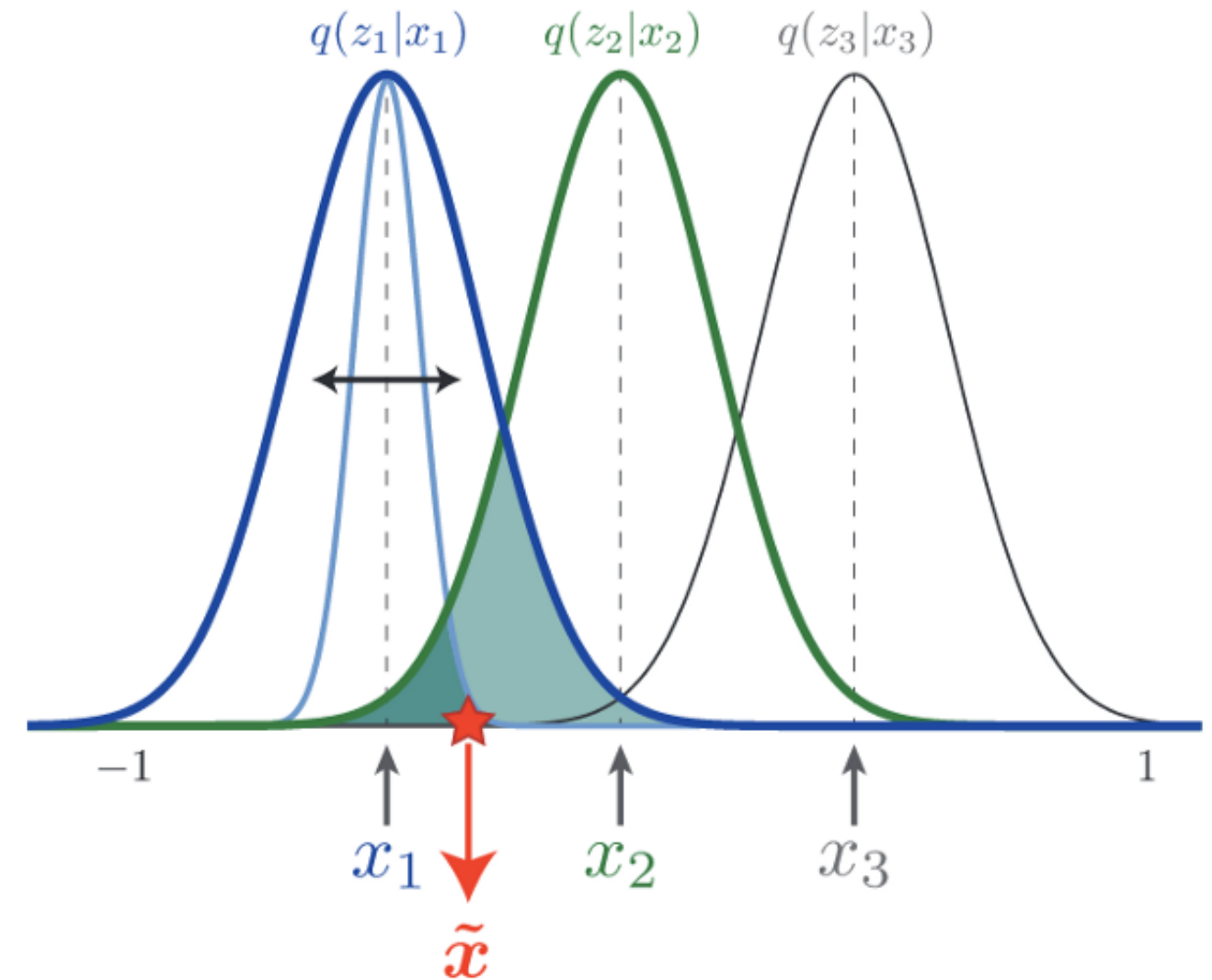
# VAE steps for MNIST

- details in [original paper](#) and notebook
- linear encoder for both  $\mu$  and  $\log(\sigma^2)$
- then transformation to  $\mathcal{N}(0, 1)$  to be able to take gradient inside expectation as in ADVI
- then decode using a loss: binary cross-entropy  $p(x|z)$  (for images) minus KL



# Disentanglement Issues

- can be understood from a gaussian mixtures perspective
- we would prefer data locality
- thus crank up the prior (regularization) term
- this is called the  $\beta$ VAE



# How to implement?

- possible in pytorch, also in pymc3
- see [convolutional VAE for MNIST in pymc3](#)
- notice that MNIST, which we did earlier as supervised is now being done unsupervised.

# Why?

See pymc3 for e.g. for [auto-encoding LDA](#)

- variational auto-encoders algorithm which allows us to perform inference efficiently for large datasets
- use tunable and flexible encoders such as multilayer perceptrons (MLPs) as our variational distribution to approximate complex variational posterior
  - then its just ADVI with mini-batch on PyMC3 or pytorch. Can use for any posterior, example LDA, or custom for MNIST