Lecture 24

VARIATIONAL INFERENCE

AM 207

Latent variables

- instead of bayesian vs frequentist, think hidden vs not hidden
- key concept: full data likelihood vs partial data likelihood
- probabilistic model is a *joint distribution* $p(\mathbf{x}, \mathbf{z})$
- observed variables $\mathbf{x}$ corresponding to data, and latent variables $\mathbf{z}$

From `edwardlib`: $p(\mathbf{x} \mid \mathbf{z})$

describes how any data $\mathbf{x}$ depend on the latent variables $\mathbf{z}$.

- **The likelihood posits a data generating process**, where the data $\mathbf{x}$ are assumed drawn from the likelihood conditioned on a particular hidden pattern described by $\mathbf{z}$.

- The *prior* $p(\mathbf{z})$ is a probability distribution that describes the latent variables present in the data. **The prior posits a generating process of the hidden structure**.

Generative Model: How to simulate from it?

$$Z \sim Categorical(\lambda_1, \lambda_2, \ldots, \lambda_K)$$

where Z says which component X is drawn from.

Thus λ_j is the probability that the hidden class variable $z = j$.

Then: $X \sim p_z(x|\theta_z)$ and general structure is:

$$p(x|\theta) = \sum_z p(x, z|\theta) = \sum_z p(z)p(x|z, \theta) \text{ where } \theta = \{\theta_k\}.$$

Concrete Formulation of unsupervised learning

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

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

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

The EM algorithm, conceptually

- iterative method for maximizing difficult likelihood (or posterior) problems, first introduced by Dempster, Laird, and Rubin in 1977
- Sorta like, just assign points to clusters to start with and iterate.
- Then, at each iteration, replace the augmented data by its conditional expectation given current observed data and parameter estimates. (E-step)
- Maximize the full-data likelihood (M-step).

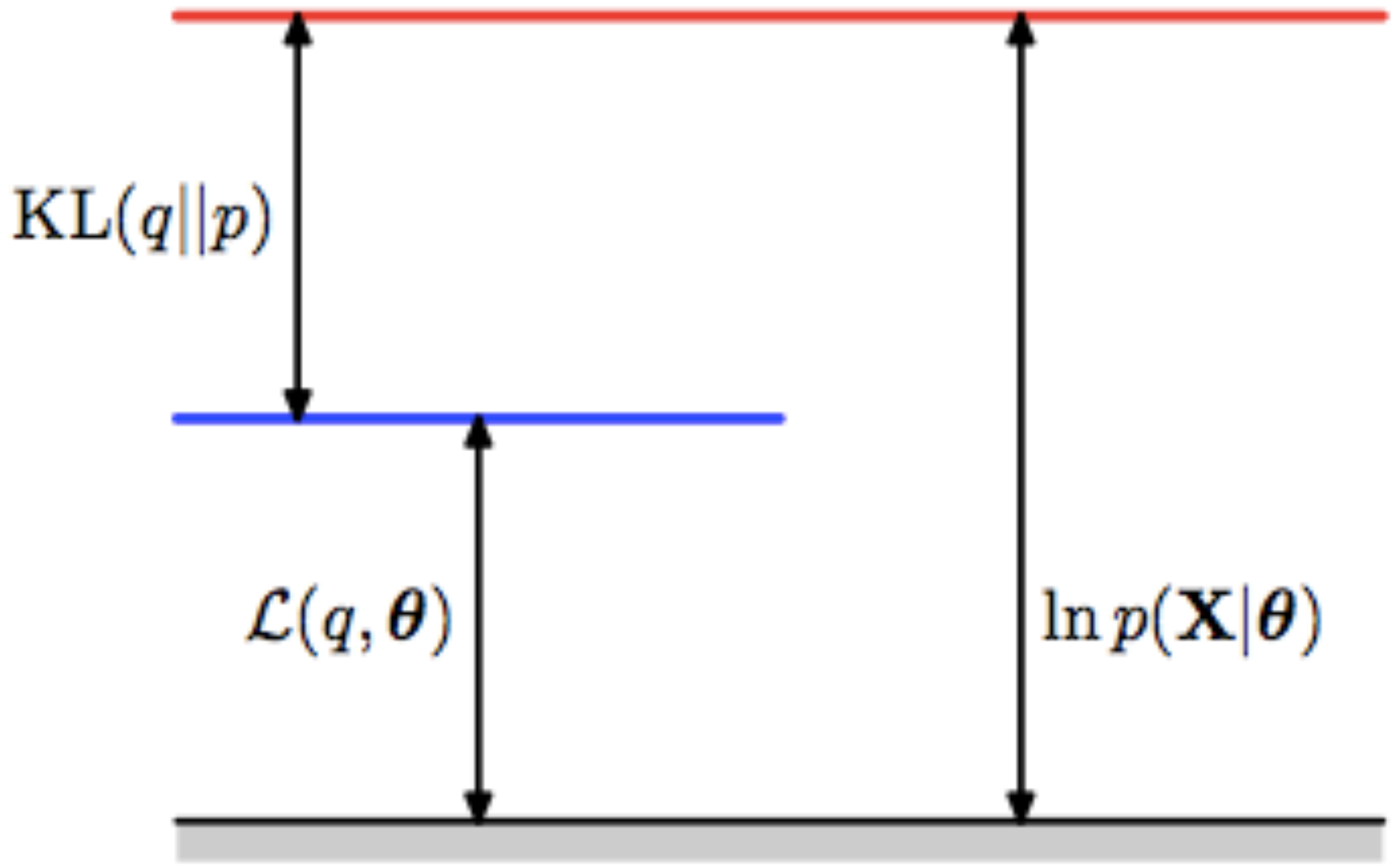
x-data likelihood

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

If we define the ELBO or Evidence Lower bound as:

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

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


$\mathrm{KL}(q||p)$
$\mathcal{L}(q,\boldsymbol{\theta})$
$\ln p(\mathbf{X}|\boldsymbol{\theta})$

- 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:

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

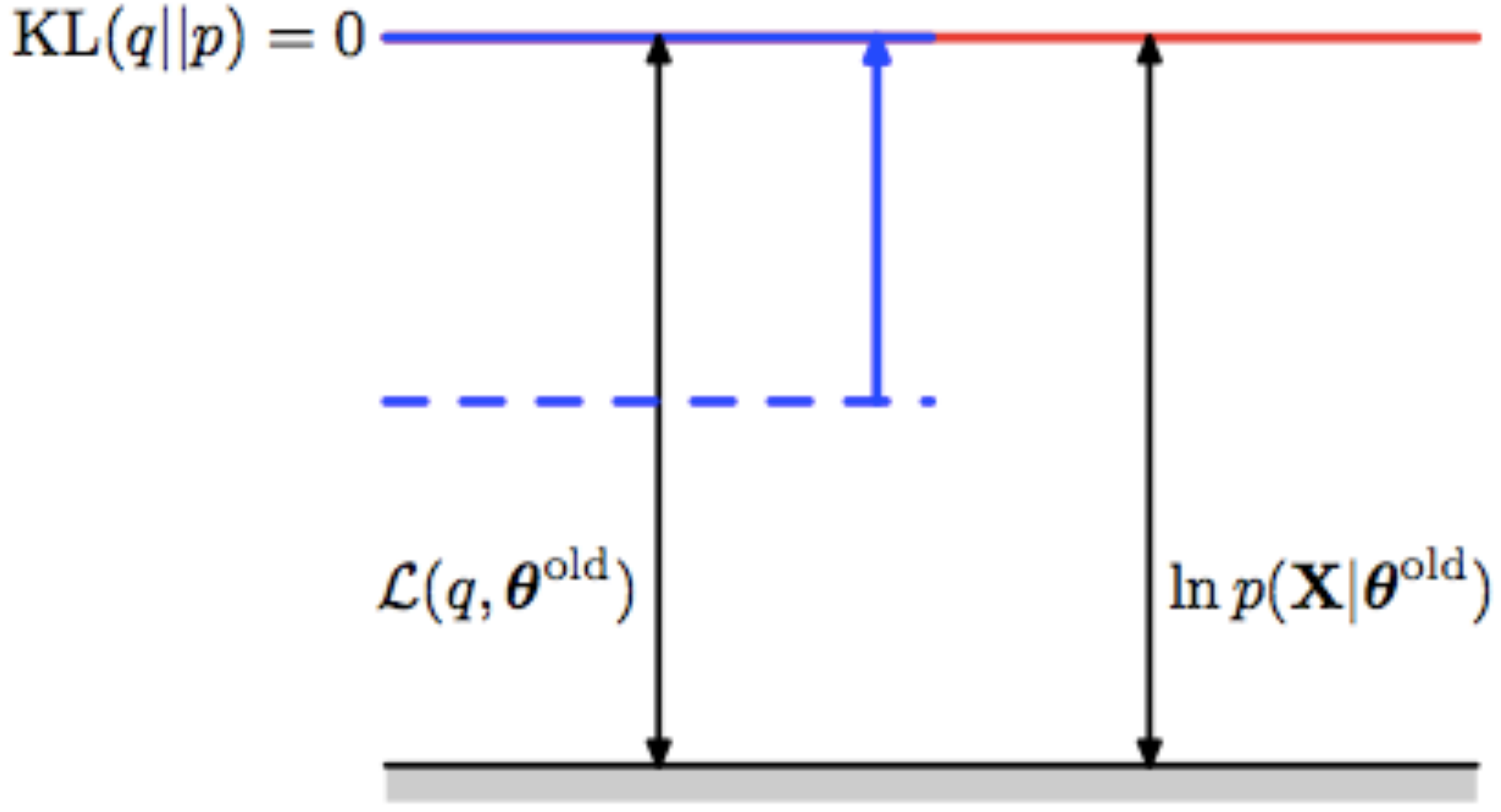
E-step conceptually

Choose at some (possibly initial) value of the parameters θ_{old},

$$q(z) = p(z|x, \theta_{old}),$$

then KL divergence = 0, and thus $\mathcal{L}(q, \theta)$ = log-likelihood at θ_{old}, maximizing the ELBO.

Conditioned on observed data, and θ_{old}, we use q to **conceptually** compute the expectation of the missing data.

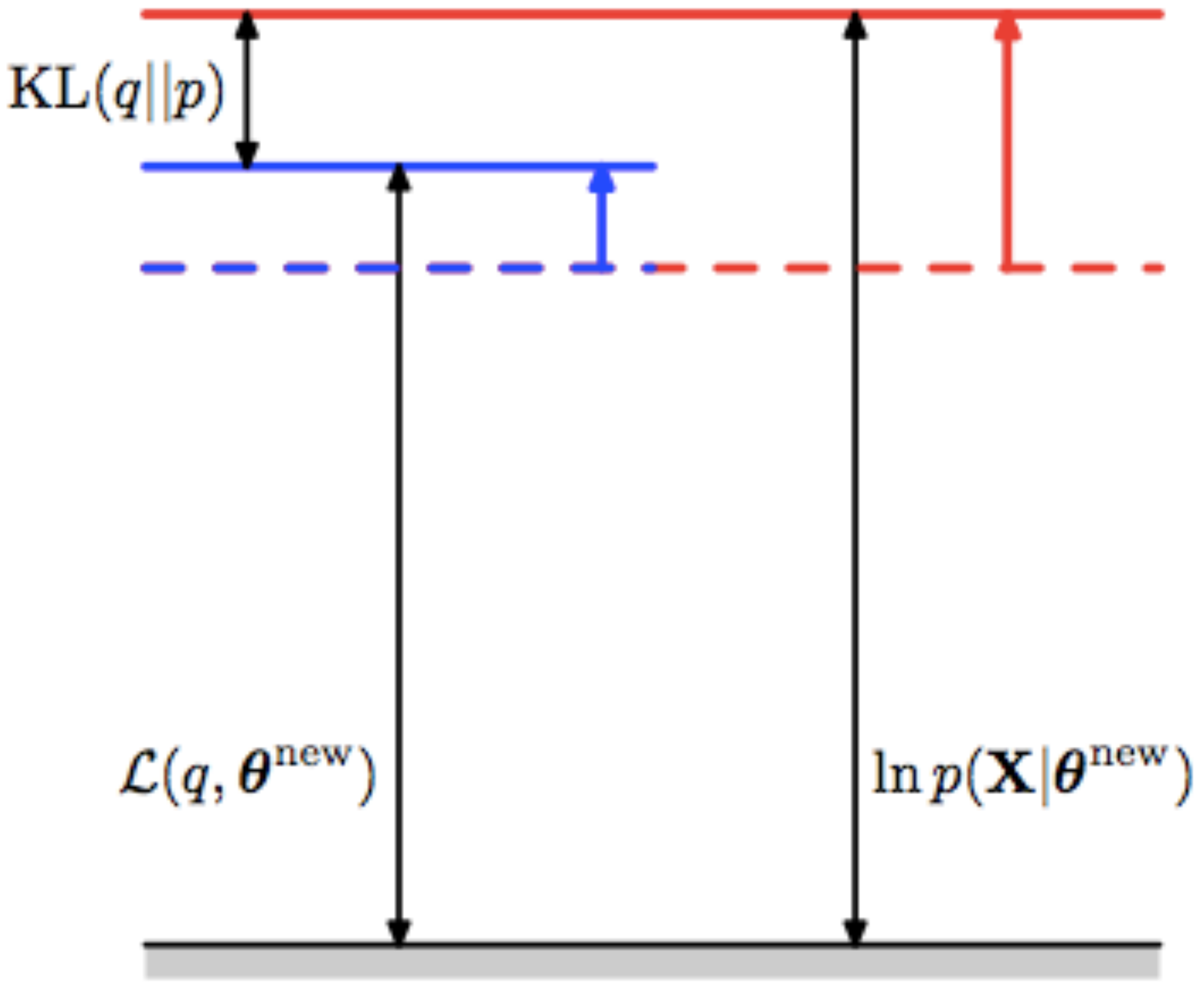

$\mathrm{KL}(q||p) = 0$
$\mathcal{L}(q, \boldsymbol{\theta}^{\mathrm{old}})$
$\ln p(\mathbf{X}|\boldsymbol{\theta}^{\mathrm{old}})$

E-step: what we actually do

Compute the Auxilary function, $Q(\theta, \theta^{(t-1)})$, the expected complete(full) data log likelihood, defined by:

$$Q(\theta, \theta^{(t-1)}) = E_{Z|Y=y, \Theta=\theta^{t-1}}[logp(x, z|\theta)]$$

or the expectation of the ELBO instead of Q.


$\mathrm{KL}(q||p)$
$\mathcal{L}(q, \boldsymbol{\theta}^{\text{new}})$
$\ln p(\mathbf{X}|\boldsymbol{\theta}^{\text{new}})$

M-step

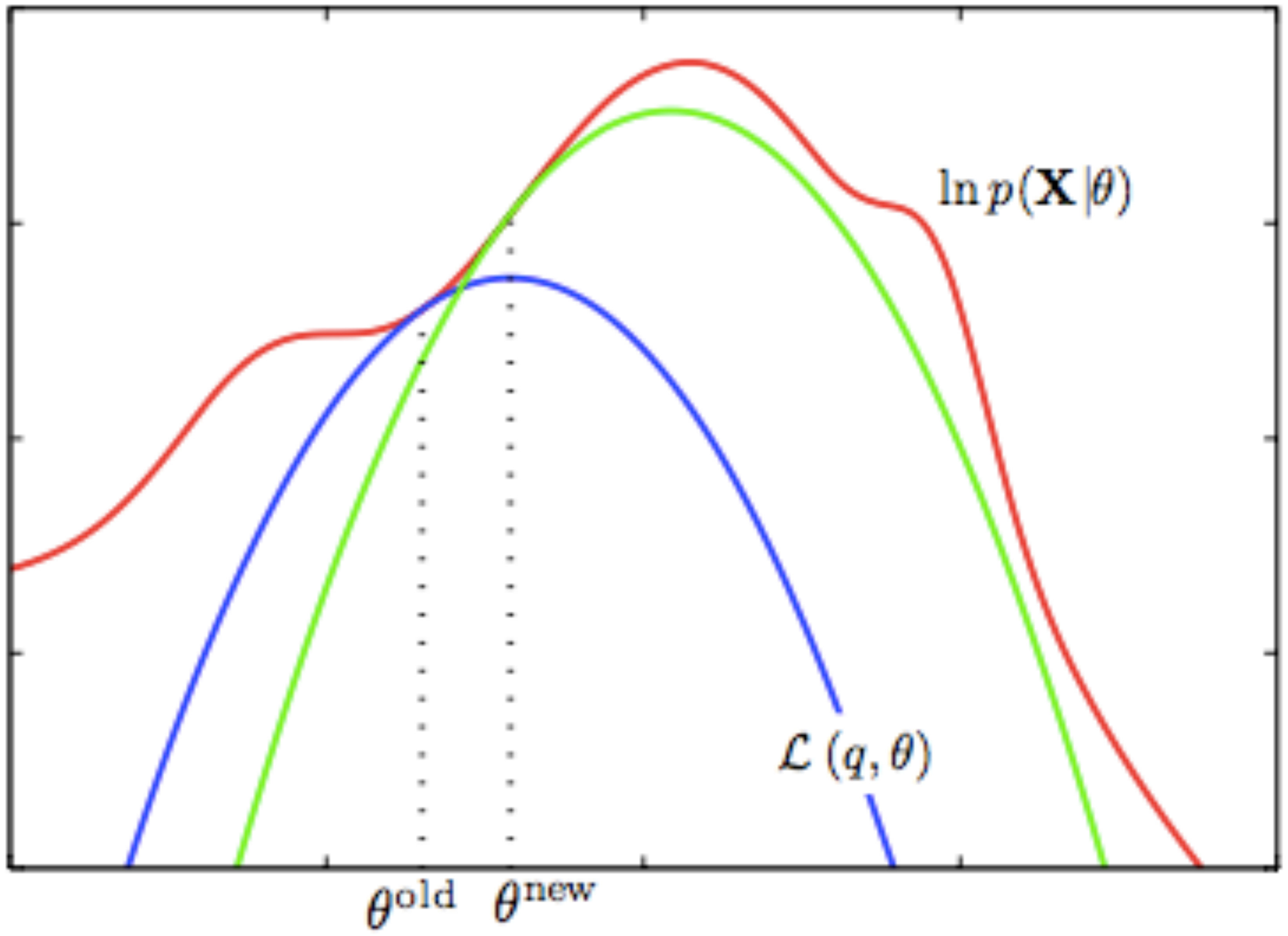
After E-step, ELBO touches $\ell(x|\theta)$, any maximization wrt θ will also “push up” on likelihood, thus increasing it.

Thus hold $q(z)$ fixed at the z-posterior calculated at θ_{old}, and maximize ELBO $\mathcal{L}(q, \theta, \theta_{old})$ or $Q(q, \theta, \theta_{old})$ wrt θ to obtain new θ_{new}.

In general $q(\theta_{old}0 \neq p(z|x, \theta_{new})$, hence KL $\neq 0$. Thus increase in $\ell(x|\theta) \geq$ increase in ELBO.

Process

1. Start with $p(x|\theta)$(red curve), θ_{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 θ_{old}.
    2. M-step: maximize Q or $ELBO$ wrt θ to get θ_{new}.
    3. Set $\theta_{old} = \theta_{new}$



GMM

E-step: Calculate $w_{i,j} = q_i(z_i = j) = p(z_i = j | x_i, \lambda, \mu, \Sigma)$

M-step: maximize: $\mathcal{L} = \sum_i \sum_{z_i} q_i(z_i) \log \frac{p(x_i, z_i | \lambda, \mu, \Sigma)}{q_i(z_i)}$

$$\mathcal{L} = \sum_i \sum_{j=i}^{k} q_i(z_i = j) \log \frac{p(x_i | z_i = j, \mu, \Sigma) p(z_i = j | \lambda)}{q_i(z_i = j)}$$

$$\mathcal{L} = \sum_{i=1}^{m} \sum_{j=i}^{k} w_{i,j} \log \left[\frac{\frac{1}{(2\pi)^{n/2} |\Sigma_j|^{1/2}} \exp\left(-\frac{1}{2} (x_i - \mu_j)^T \Sigma_j^{-1} (x_i - \mu_j) \right) \lambda_j}{w_{i,j}} \right]$$

M-step

Taking derivatives yields following updating formulas:

$$\lambda_j = \frac{1}{m} \sum_{i=1}^{m} w_{i,j}$$

$$\mu_j = \frac{\sum_{i=1}^{m} w_{i,j}\, x_i}{\sum_{i=1}^{m} w_{i,j}}$$

$$\Sigma_j = \frac{\sum_{i=1}^{m} w_{i,j}\, (x_i - \mu_j)(x_i - \mu_j)^T}{\sum_{i=1}^{m} w_{i,j}}$$

E-step: calculate responsibilities

We are basically calculating the posterior of the z's given the x's and the current estimate of our parameters. We can use Bayes rule

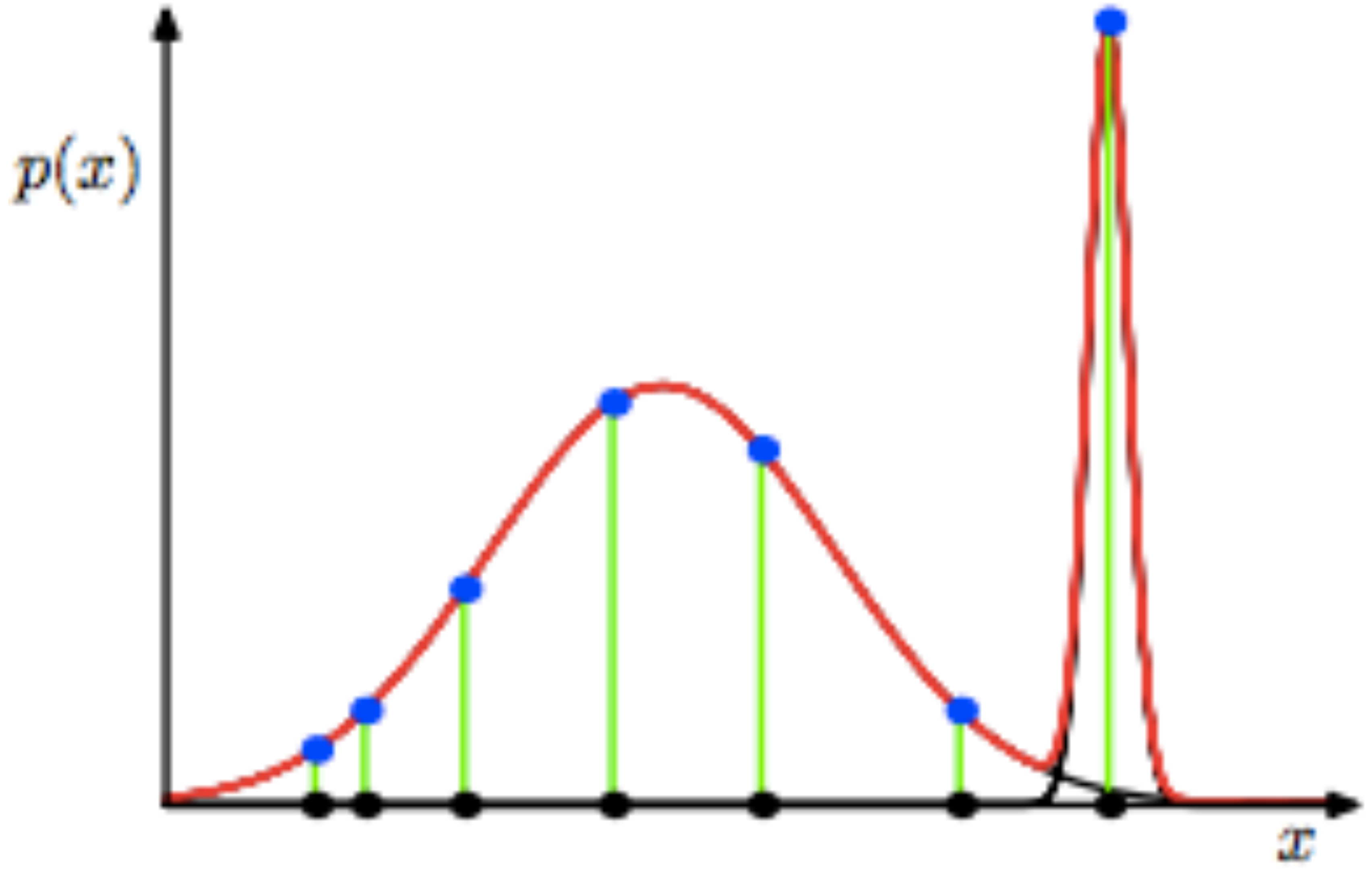
$$w_{i,j} = p(z_i = j|x_i, \lambda, \mu, \Sigma) = \frac{p(x_i|z_i = j, \mu, \Sigma)\, p(z_i = j|\lambda)}{\sum_{l=1}^{k} p(x_i|z_i = l, \mu, \Sigma)\, p(z_i = l|\lambda)}$$

Where $p(x_i|z_i = j, \mu, \Sigma)$ is the density of the Gaussian with mean μ_j and covariance Σ_j at x_i and $p(z_i = j|\lambda)$ is simply λ_j.

Compared to supervised classification and k-means

- M-step formulas vs GDA we can see that are very similar except that instead of using δ functions we use the w's.
- Thus the EM algorithm corresponds here to a weighted maximum likelihood and the weights are interpreted as the 'probability' of coming from that Gaussian
- Thus we have achieved a **soft clustering** (as opposed to k-means in the unsupervised case and classification in the supervised case).

- kmeans is HARD EM. Instead of calculating Q in e-step, use mode of z posterior. Also the case with classification
- finite mixture models suffer from multimodality, non-identifiability, and singularity. They are problematic but useful
- models can be singular if cluster has only one data point: overfitting
- add in prior to regularise and get MAP. Add log(prior) in M-step only



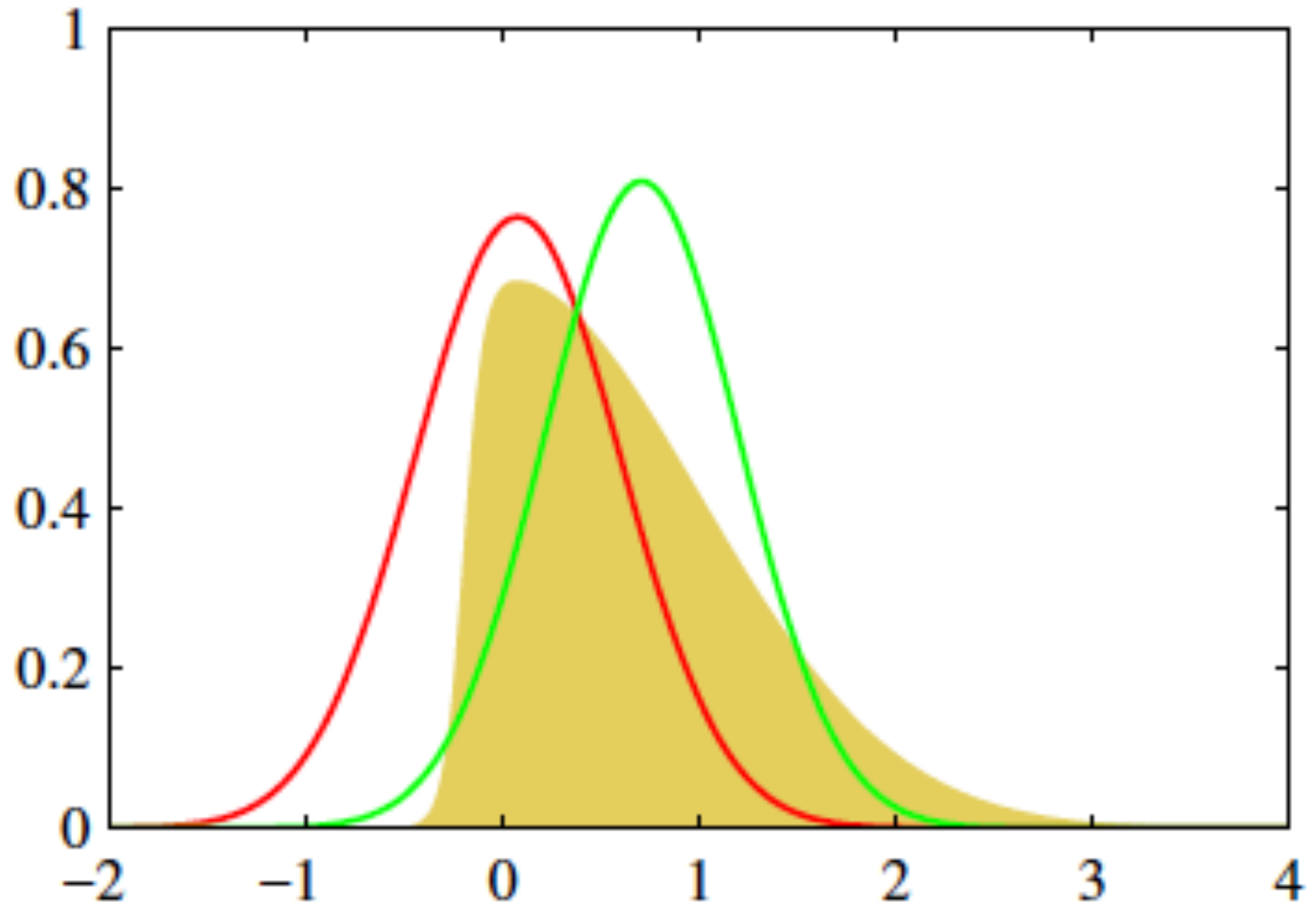
VARIATIONAL INFERENCE

Core Idea

z is now all parameters. Dont distinguish from θ.

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:

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



VI vs MCMC

MCMC	**VI**
More computationally intensive	Less intensive
Guarantees producing asymptotically exact samples from target distribution	No such guarantees
Slower	Faster, especially for large data sets and complex distributions
Best for precise inference	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)

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

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

(likelihood-prior balance)

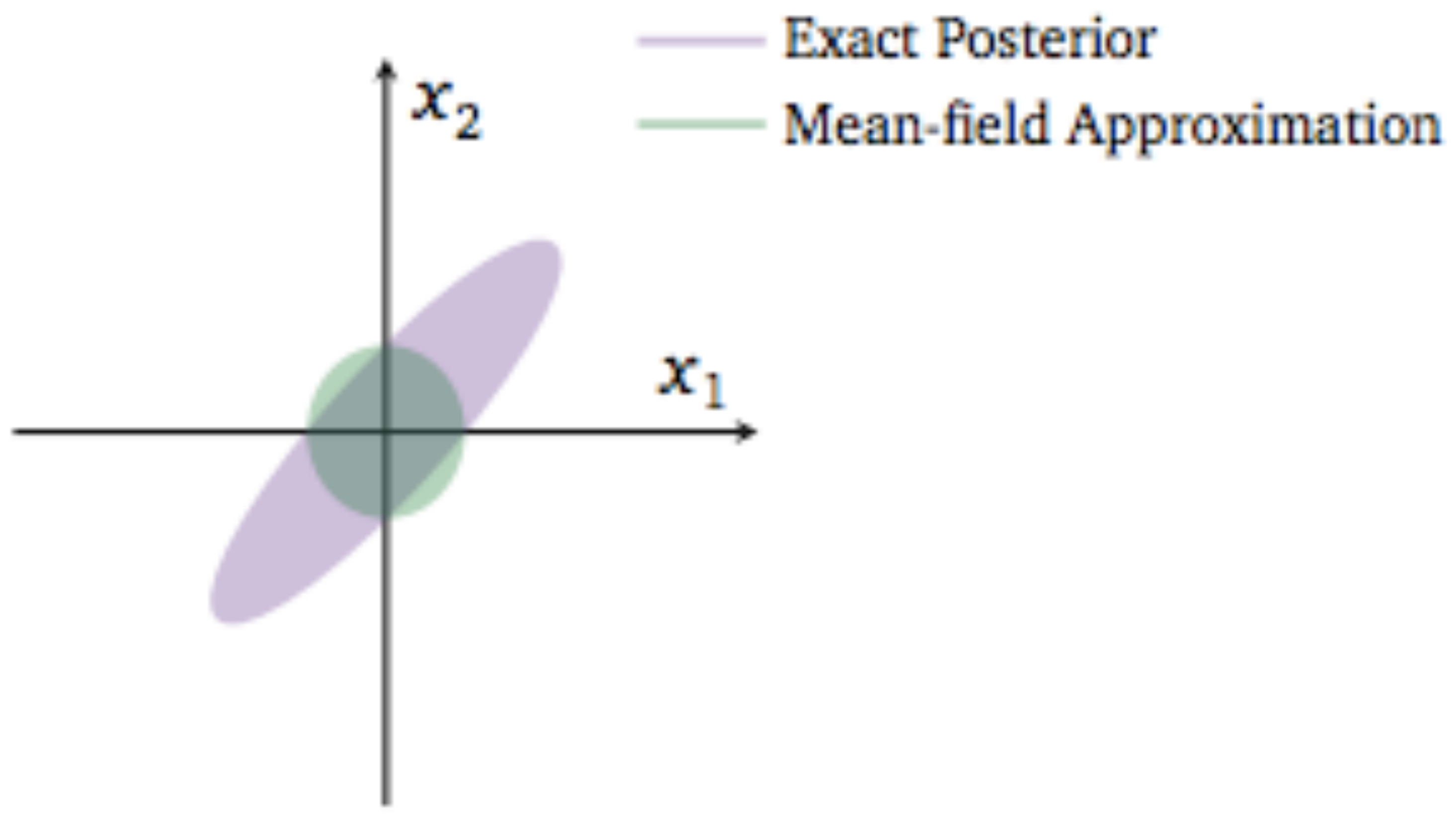
Mean Field: Find a q such that:

$KL + ELBO = log(p(x))$: 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 $z_j = p(z_j | \boldsymbol{z_{-j}}, \boldsymbol{x})$

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:

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

    compute ELBO
```

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

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

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

(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(sd)`. So, for e.g.,

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

For the second term below, we have only retained what depends on $q_j(z_j)$

$$ELBO(q) = E_q[(log(p(z,x))] - E_q[log(q(z))]$$
$$\implies ELBO(q_j) = E_j[E_{-j}[log(p(z_j, \boldsymbol{z_{-j}}, \boldsymbol{x}))]] - E_j[log(q_j(z_j))] + constants$$
$$\implies ELBO(q_j) = E_j[A] - E_j[log(q_j(z_j))] + constants$$

Upto an added constant, $RHS = -D_{KL}(q_j, exp(A))$. Thus, maximizing $ELBO(q_j)$ same as minimizing KL divergence.

This occurs when $q_j = exp(A)$. Thus CAVI locally maximizes ELBO.

Example: Gaussian Mixture Model

$$\boldsymbol{\mu} = \{\mu_1, \ldots, \mu_K\}$$

$$\mu_k \sim \mathcal{N}(0, \sigma^2),\ \ k = 1, \ldots, K$$

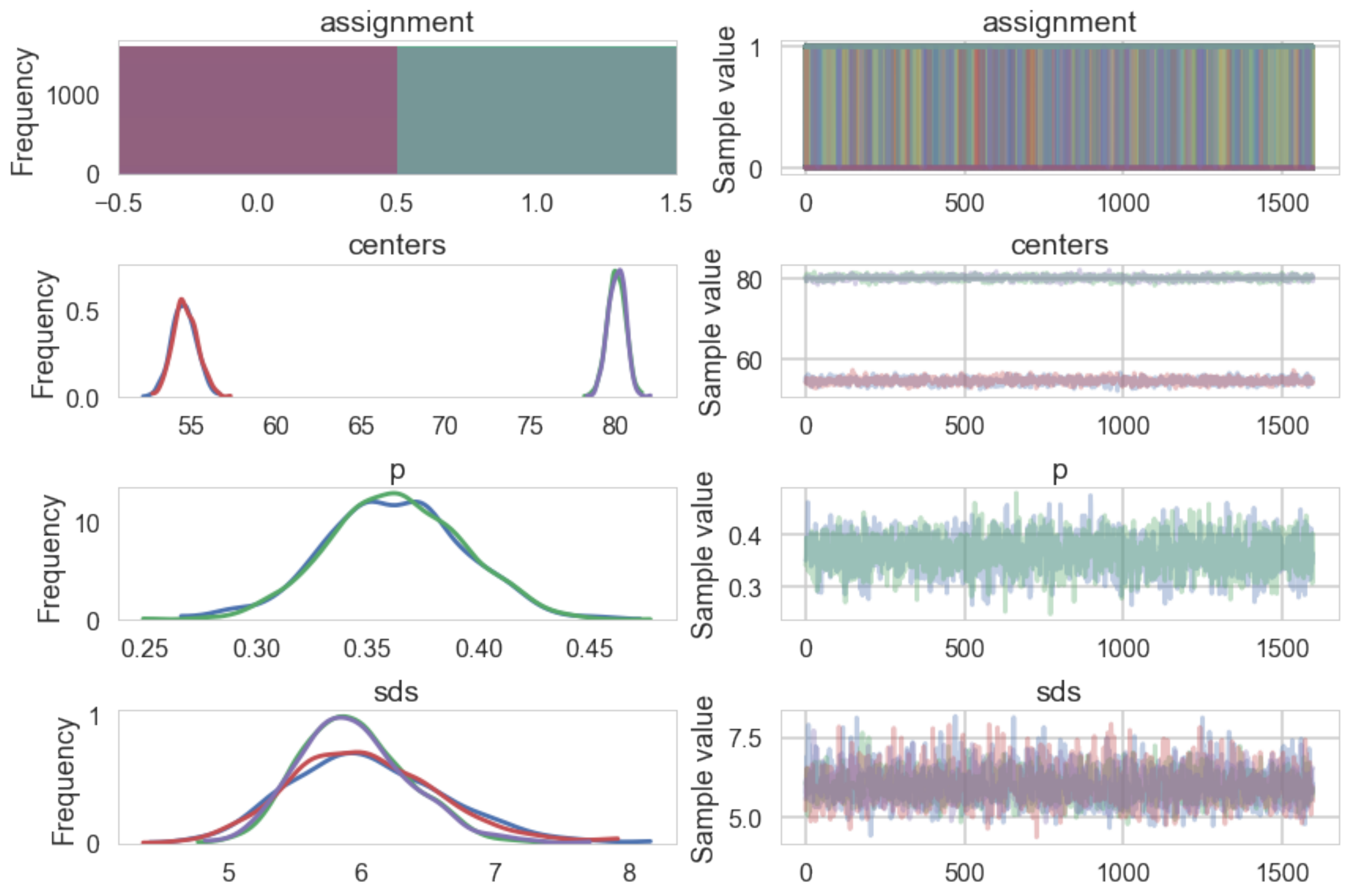
$$c_i \sim Categorical(\frac{1}{K}, \ldots, \frac{1}{K}),\ \ i = 1, \ldots, n\ (c_i)$$

$$x_i | c_i, \boldsymbol{\mu} \sim \mathcal{N}(c_i^T \boldsymbol{\mu}, 1),\ \ i = 1, \ldots, n$$

Sampling mixture models: 2 Gaussians

```
with pm.Model() as ofmodel:
    p1 = pm.Uniform('p', 0, 1)
    p2 = 1 - p1
    p = tt.stack([p1, p2])
    assignment = pm.Categorical("assignment", p,
                                shape=ofdata.shape[0])
    sds = pm.Uniform("sds", 0, 40, shape=2)
    centers = pm.Normal("centers",
                mu=np.array([50, 80]),
                sd=np.array([20, 20]),
                shape=2)

    observations = pm.Normal("obs",
        mu=centers[assignment],
        sd=sds[assignment],
        observed=ofdata.waiting)
```



Full data joint: $p(\boldsymbol{\mu}, \boldsymbol{c}, \boldsymbol{x}) = p(\boldsymbol{\mu}) \prod_{\boldsymbol{i=1}}^{\boldsymbol{n}} p(c_i)p(x_i|c_i, \boldsymbol{\mu})$

Evidence: $p(\boldsymbol{x}) = \int d\boldsymbol{\mu} p(\boldsymbol{\mu}) \prod_{i=1}^{n} \sum_{c_i} p(c_i)p(x_i|c_i, \boldsymbol{\mu})$

This integral does not reduce to a product of 1-d integrals for each of the μs. Evidence as usual hard to compute.

The latent variables are the K class means and the n class assignments - $\boldsymbol{z} = \{\boldsymbol{\mu}, \boldsymbol{c}\}$ (thats why we marginalize in MCMC)

Mean-field Variational Family

$$q(\boldsymbol{\mu}, \boldsymbol{c}) = \prod_{k=1}^{K} q(\mu_k ; m_k , s_k^2) \prod_{i=1}^{n} q(c_i ; w_i)$$

- First factor: Gaussian distribution on the kth mixture component's mean, parameterized by its own mean and variance
- Second factor: ith observation's mixture assignment with assignment probabilities given by a K-vector w_i, and c_i being the bit-vector (with one 1) associated with data point i.

ELBO

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

$$\implies ELBO(\boldsymbol{m}, \boldsymbol{s^2}, \boldsymbol{w}) = \sum_{k=1}^{K} E_q[log(p(\mu_k)); m_k, s_k^2] \text{ (Q..)}$$

$$+ \sum_{i=1}^{n} (E_q[log(p(c_i)); w_i] + E_q[log(p(x_i | c_i, \boldsymbol{\mu})); w_i, \boldsymbol{m}, \boldsymbol{s^2}]) \text{ (..Q)}$$

$$- \sum_{i=1}^{n} E_q[log(q(c_i; w_i))] - \sum_{k=1}^{K} E_q[log(q(\mu_k; m_k, s_k^2))] \text{ (entropy)}$$

CAVI updates: cluster assignment

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

Since we are talking about the assignment of the ith point, we can drop all points $j \neq i$ and terms for the k means.

$$\implies q^*(c_i; w_i) \propto \exp\{log(p(c_i)) + E_{-z_i}[log(p(x_i|c_i, \boldsymbol{\mu})); \boldsymbol{m}, \boldsymbol{s^2}]\}$$

$$log(p(c_i)) = log(\frac{1}{K}), p(x_i|c_i, \boldsymbol{\mu}) = \prod_{k=1}^{K} p(x_i|\mu_k)^{c_{ik}}$$

$$E_{-z_i}[log(p(x_i|c_i, \boldsymbol{\mu}))] = \sum_k c_{ik} E_{-z_i}[log(p(x_i|\mu_k)); m_k, s_k^2]$$

$$E_{-z_i}[log(p(x_i|c_i, \boldsymbol{\mu}))] = \sum_k c_{ik} E_{-z_i}[-0.5(x_i - \mu_k)^2; m_k, s_k^2] + C$$

$$E_{-z_i}[log(p(x_i|c_i, \boldsymbol{\mu}))] = \sum_k c_{ik} (E_{-z_i}[\mu_k; m_k, s_k^2] x_i - E_{-z_i}[\mu_k^2; m_k, s_k^2]/2) + C$$

where C are constants. Substituting back into the first equation and removing terms that are constant with respect to c_i, we get the final CAVI update below.

$$w_{ik} = q^*(z_i = k) \propto \exp\{E_{-z_i}[\mu_k; m_k, s_k^2]x_i - E_{-z_i}[\mu_k^2; m_k, s_k^2]/2\}$$

As is evident, the update is purely a function of the other variational factors and can thus be easily computed.

CAVI updates: kth mixture component mean

Intuitively, these posteriors are gaussian as the conditional distribution of μ_k is a gaussian with the data being the data "assigned" to the kth cluster.

Note that since c_i is an indicator vector:

$$w_{ik} = E_{-\mu_k}[c_{ik}; w_i]$$

$$log(q(\mu_k)) = log(p(\mu_k)) + \sum_i E_{-\mu_k}[log(p(x_i|c_i, \boldsymbol{\mu})); w_i, \boldsymbol{m_{-k}}, \boldsymbol{s^2_{-k}}] + C$$

$$\implies log(q(\mu_k)) = log(p(\mu_k)) + \sum_i E_{-\mu_k}[c_{ik} log(p(x_i|\mu_k)); w_i] + C$$

$$\implies log(q(\mu_k)) = -\mu_k^2/2\sigma^2 + \sum_i E_{-\mu_k}[c_{ik}; w_i] log(p(x_i|\mu_k)) + C$$

$$\implies log(q(\mu_k)) = -\mu_k^2/2\sigma^2 + \sum_i w_{ik}(-(x_i - \mu_k)^2/2) + C$$

$$\implies log(q(\mu_k)) = -\mu_k^2/2\sigma^2 + \sum_i (w_{ik} x_i \mu_k - w_{ik}\mu_k^2/2) + C$$

$$\implies log(q(\mu_k)) = (\sum_i w_{ik} x_i)\mu_k - (1/2\sigma^2 + \sum_i w_{ik}/2)\mu_k^2 + C$$

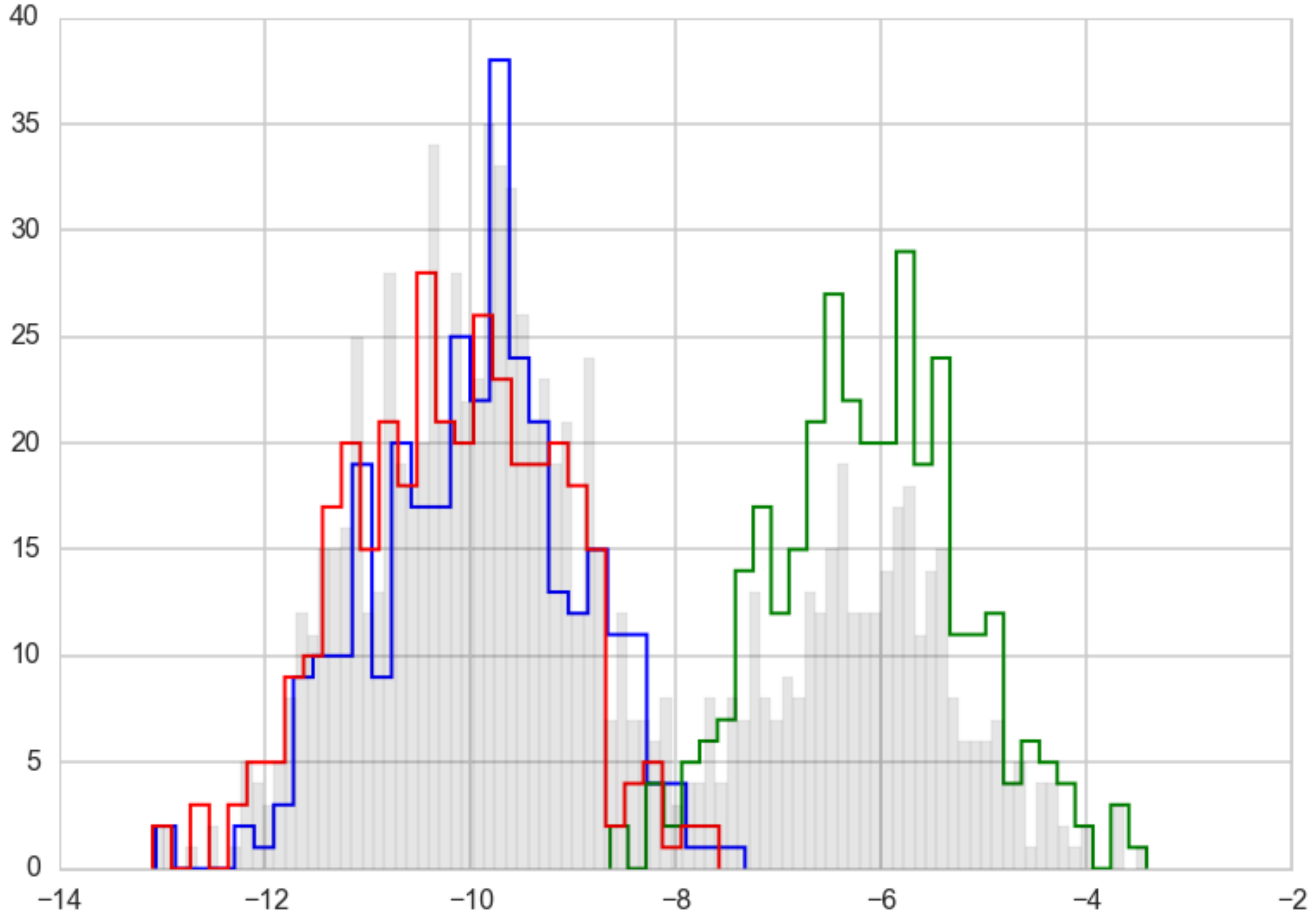
$$\implies q(\mu_k) = Gaussian$$

$$m_k = \frac{\sum_i w_{ik} x_i}{1/\sigma^2 + \sum_i w_{ik}}$$

$$s_k^2 = \frac{1}{1/\sigma^2 + \sum_i w_{ik}}$$

CAVI update for the kth mixture component takes the form of a Gaussian distribution parameterized by the above derived mean and variance.

3 gaussian mixture in code



```python
n = 1000
# hyperparameters
prior_std = 10

# True parameters
K = 3
mu = []
for i in range(K):
    mu.append(np.random.normal(0, prior_std))

var = 1
var_arr = [1, 1, 1]

# Run the CAVI algorithm
mixture_components, c_est = VI(K, prior_std, n, data)
```

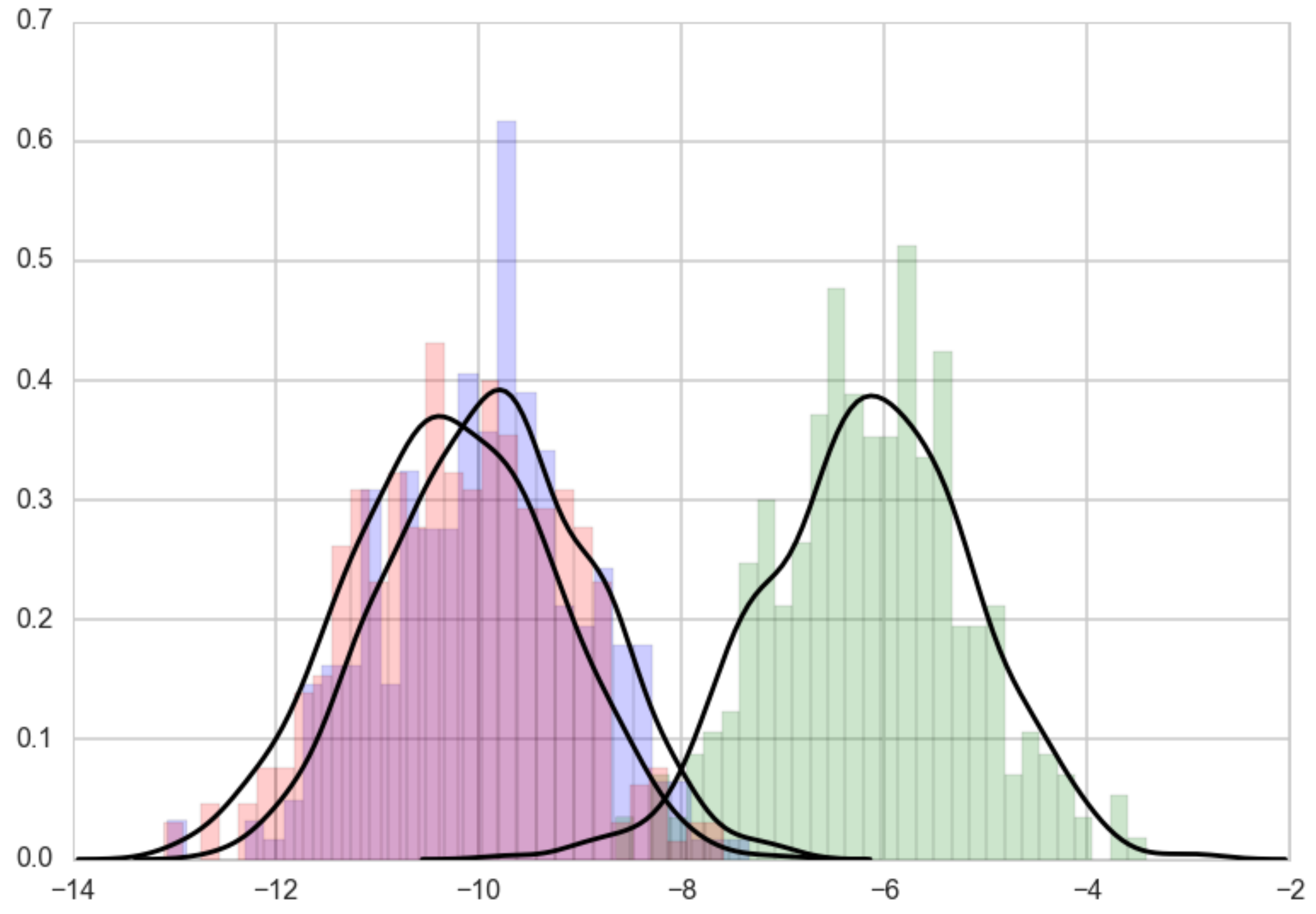
```
def VI(K, prior_std, n, data): #VI with CAVI
    # Initialization
    mu_mean = []
    mu_var = []
    for i in range(K):
        mu_mean.append(np.random.normal(0, prior_std))
        mu_var.append(abs(np.random.normal(0, prior_std)))
    c_est = np.zeros((n, K))
    for i in range(n):
        c_est[i, np.random.choice(K)] = 1

    # Initiate CAVI iterations
    while(True):
        mu_mean_old = mu_mean[:] #copy
        # mixture model parameter update step
        for j in range(K):
            nr = 0
            dr = 0
            for i in range(n):
                nr += c_est[i, j]*data[i]
                dr += c_est[i, j]
            mu_mean[j] = nr/((1/prior_std**2) + dr)
            mu_var[j] = 1.0/((1/prior_std**2) + dr)

        # categorical vector update step
        for i in range(n):
            cat_vec = []
            for j in range(K):
                cat_vec.append(math.exp(mu_mean[j]*data[i] - (mu_var[j] + mu_mean[j]**2)/2))
            for k in range(K):
                c_est[i, k] = cat_vec[k]/np.sum(np.array(cat_vec))

        # check for convergence of variational factors
        diff = np.array(mu_mean_old) - np.array(mu_mean)
        if np.dot(diff, diff) < 0.000001:
            break

    # sort in ascending order
    mixture_components = list(zip(mu_mean, mu_var))
    mixture_components.sort()
    return mixture_components, c_est
```



Practical Considerations

1) The output can be sensitive to initialization values and thus iterating multiple times to find a relatively good local optimum is a good strategy
2) Look out for numerical stability issues - quite common when dealing with tiny probabilities
3) Ensure algorithm converges before using the result

ADVI

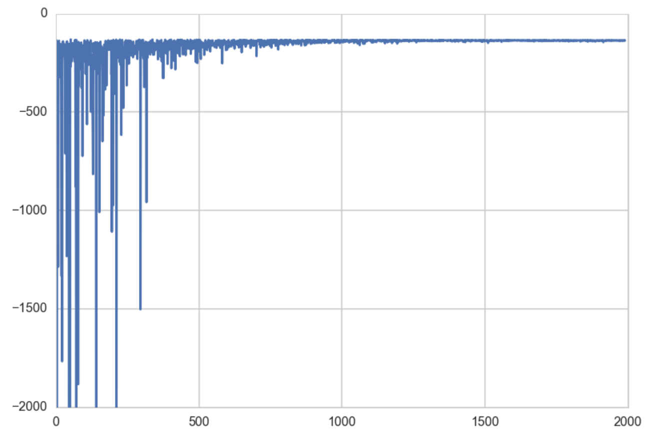
Core Idea:

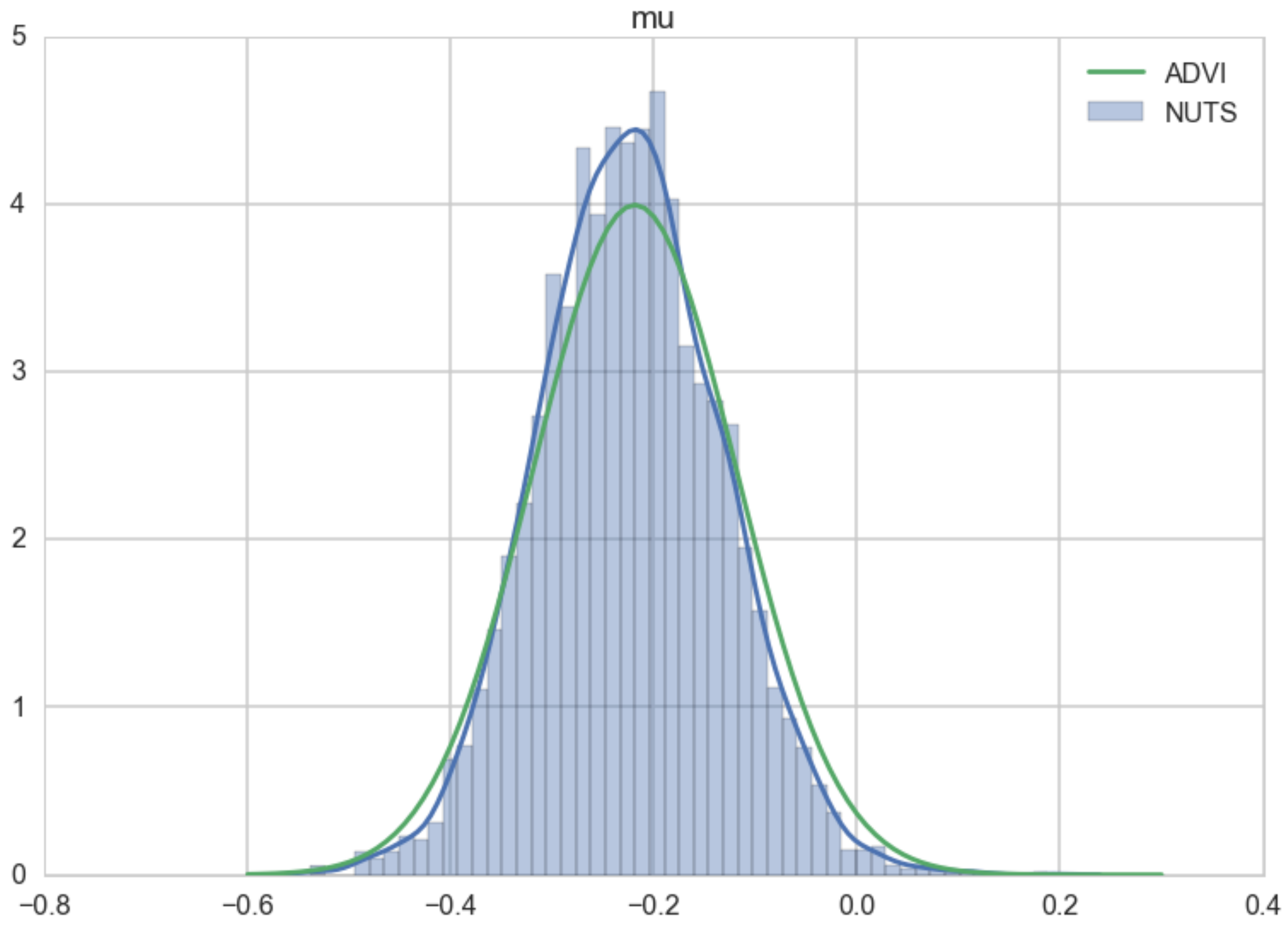
- CAVI does not scale
- Use gradient based optimization, do it on less data
- do it automatically

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]);
```

elbo:





Problem with CAVI

- does not scale
- ELBO must be painstakingly calculated
- optimized with custom CAVI updates for each new model
- If you choose to use a gradient based optimizer then you must supply gradients.

ADVI solves this problem automatically. The user specifies the model, expressed as a program, and ADVI automatically generates a corresponding variational algorithm. The idea is to first automatically transform the inference problem into a common space and then to solve the variational optimization. Solving the problem in this common space solves variational inference for all models in a large class.

-ADVI Paper

What does ADVI do?

1. Transformation of latent parameters
2. Standardization transform for posterior to push gradient inside expectation
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

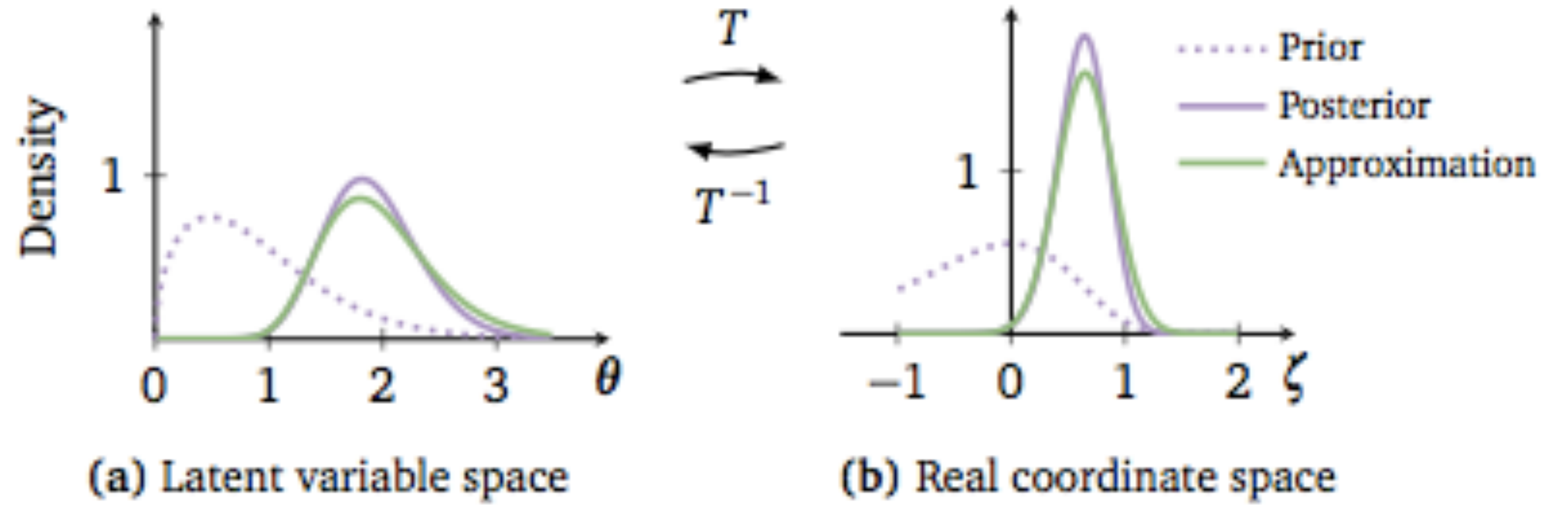
$$\nabla_\eta \mathcal{L} = E[\nabla_\eta[logp(x, T^{-1}(S^{-1}(\eta))) + log(det(J_{T^{-1}}(S^{-1}(\eta))))]$$

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

(1) S-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 $p(z, \eta)$ where η 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 σ params by simply taking their logs.



(a) Latent variable space (b) Real coordinate space

(2) T-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 tp 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.

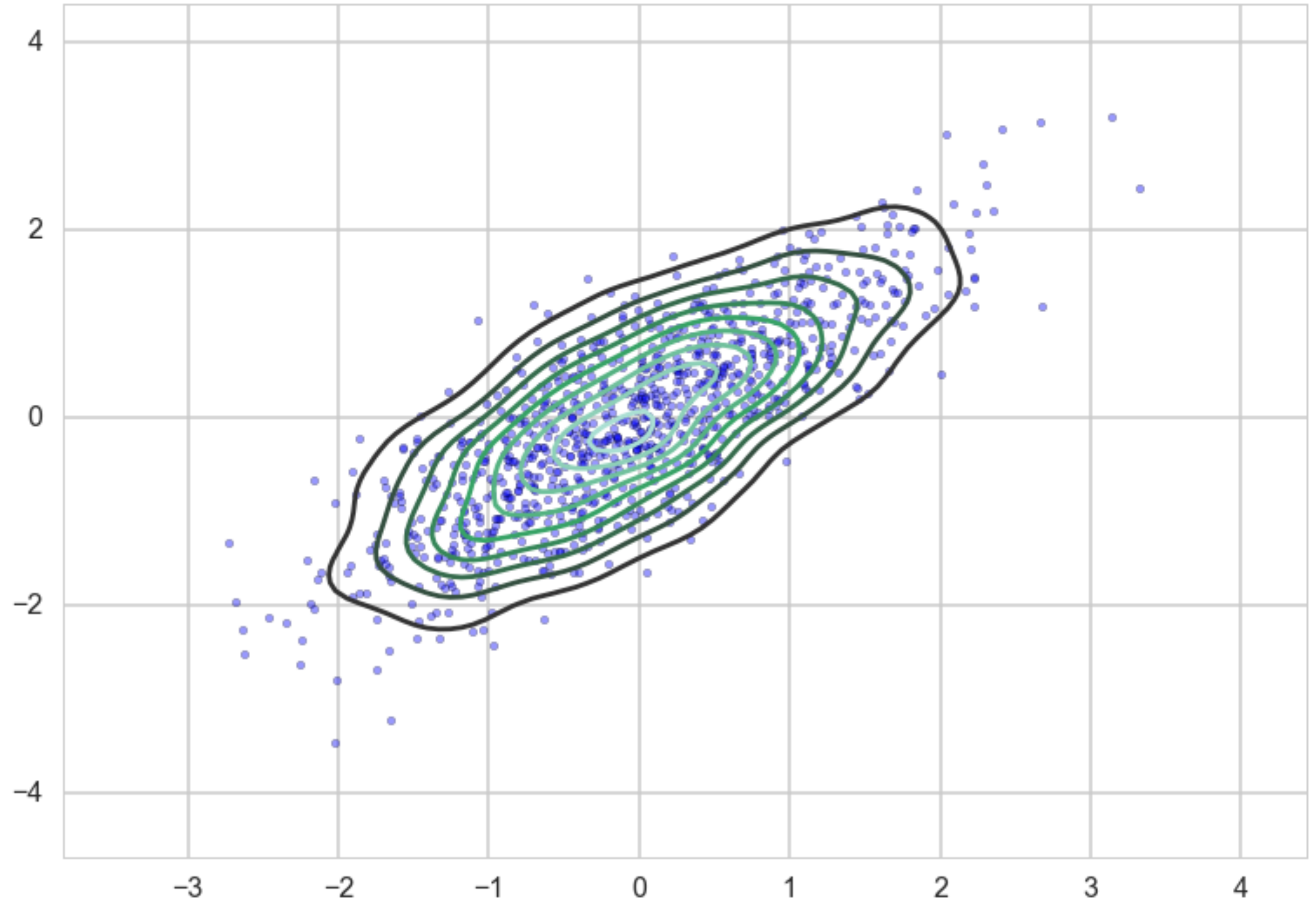
(4) Calculate the gradients

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.

Example with Mixtures in lab. Also see pymc docs for variational ANN and autoencoders.

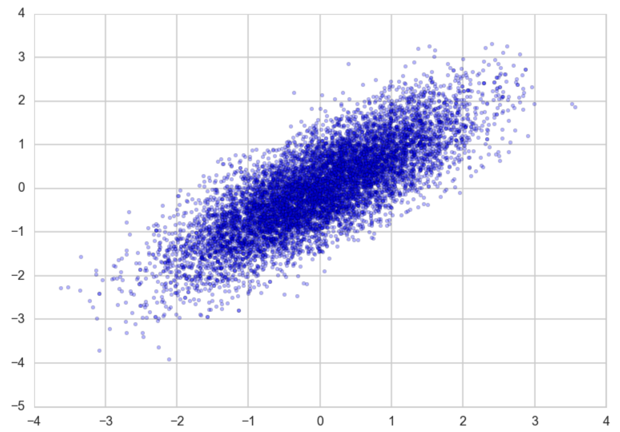
2D gaussian example



High correlation gaussian with sampler

```
cov=np.array([[0,0.8],[0.8,0]], dtype=np.float64)
data = np.random.multivariate_normal([0,0], cov, size=1000)
sns.kdeplot(data);
with pm.Model() as mdensity:
    density = pm.MvNormal('density', mu=[0,0],
    cov=tt.fill_diagonal(cov,1), shape=2)
with mdensity:
    mdtrace=pm.sample(10000)
```

Trace:



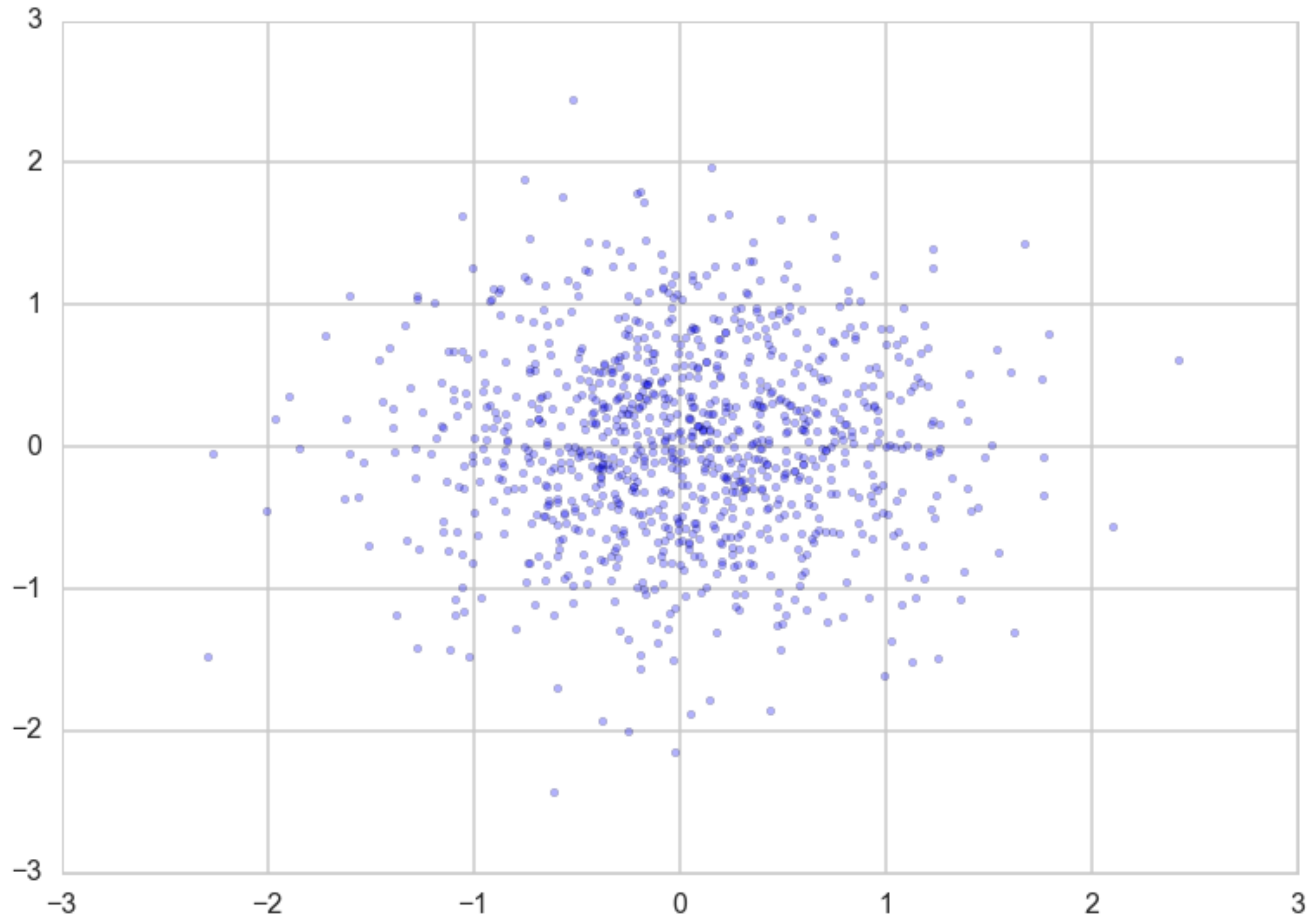
Sampling with ADVI

```
mdvar = pm.ADVI(model=mdensity)
mdvar.fit(n=40000)
samps=mdvar.approx.sample(5000)
plt.scatter(samps['density'][:,0],
    samps['density'][:,1], s=5, alpha=0.3)
```

ADVI cannot find the correlational structure.

Transform to de-correlate to use ADVI.

You have been doing this for NUTS anyways.



Where is the Variational

- variational calculus is the differentiation of functionals (functions of functions) with respect to functions
- Principles of least time in optics and least action in Physics are great examples. Also basis for path-integral formulation of quantum mechanics
- here we differentiate KL-divergence (or ELBO) with respect to q
- we do the same thing in the E-step of EM!