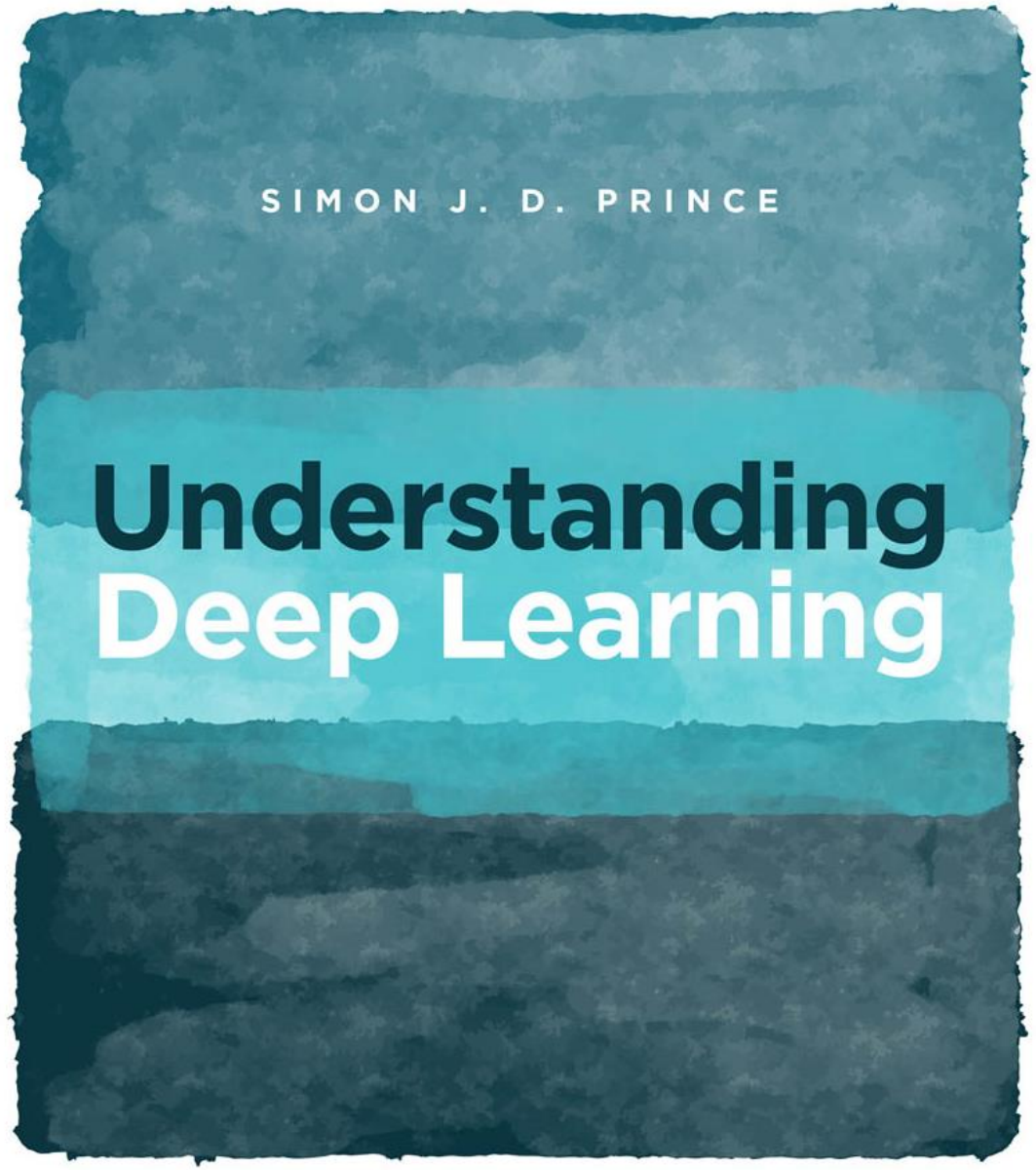

SIMON J. D. PRINCE
Understanding
Deep Learning

Understanding Deep Learning Variational Autoencoders (VAEs)

May 25, 2024

VAE chapter contents

By end of chapter, get to VAE builds a nonlinear latent variable model using an autoencoder architecture and sampling to estimate a lower bound for the likelihood

- Latent variable models
- Nonlinear latent variable model
- Training
- ELBO properties
- Variational approximation
- The variational autoencoder
- The reparameterization trick
- Applications

VAE overview

- We are going build useful, complex latent variable models
  - We will use a neural network to model our complex nonlinear function
- But maximum likelihood training with this complex latent will be computationally intractable, so we will use a series of tricks
- Instead of directly maximizing the likelihood, we will devise a lower bound for the likelihood (the ELBO) and maximize it instead
- Even the ELBO will require a variational approximation, data sampling, and a reparameterization trick to finally make all this computable
- We will end up with an encoder and a decoder, but this will be very different from a regular autoencoder because we will have a probability distribution in the middle instead of a fixed embedding

Latent variable models

- We want to know a probability distribution *Pr*(***x***)
- Instead of directly pursuing, model a joint distribution *Pr*(***x***, ***z***)
  - The latent variable **z** is unobserved
- Our *Pr*(***x***) can be found by marginalizing out ***z***:

$$Pr(\mathbf{x}) = \int Pr(\mathbf{x}, \mathbf{z}) d\mathbf{z}. \qquad (17.1)$$

- We can rewrite the above using *Pr*(***x***|***z***) = *Pr*(***x***, ***z***) / *Pr*(***z***)

$$Pr(\mathbf{x}) = \int Pr(\mathbf{x}|\mathbf{z}) Pr(\mathbf{z}) d\mathbf{z}. \qquad (17.2)$$

1D Mixture of Gaussians [1]

- Discrete latent variable *z* with *N* possible values, each with probability λ_n
- The probability distribution of *x* for each value of *z* is a Gaussian

$$\begin{aligned} Pr(z=n) &= \lambda_n \\ Pr(x|z=n) &= \mathrm{Norm}_x\left[\mu_n, \sigma_n^2\right]. \end{aligned} \tag{17.3}$$

- We marginalize over z by summing all possibilities (if it were continuous, we would have done integration)

$$\begin{aligned} Pr(x) &= \sum_{n=1}^{N} Pr(x, z=n) \\ &= \sum_{n=1}^{N} Pr(x|z=n) \cdot Pr(z=n) \\ &= \sum_{n=1}^{N} \lambda_n \cdot \mathrm{Norm}_x\left[\mu_n, \sigma_n^2\right]. \end{aligned} \tag{17.4}$$

1D Mixture of Gaussians [2]

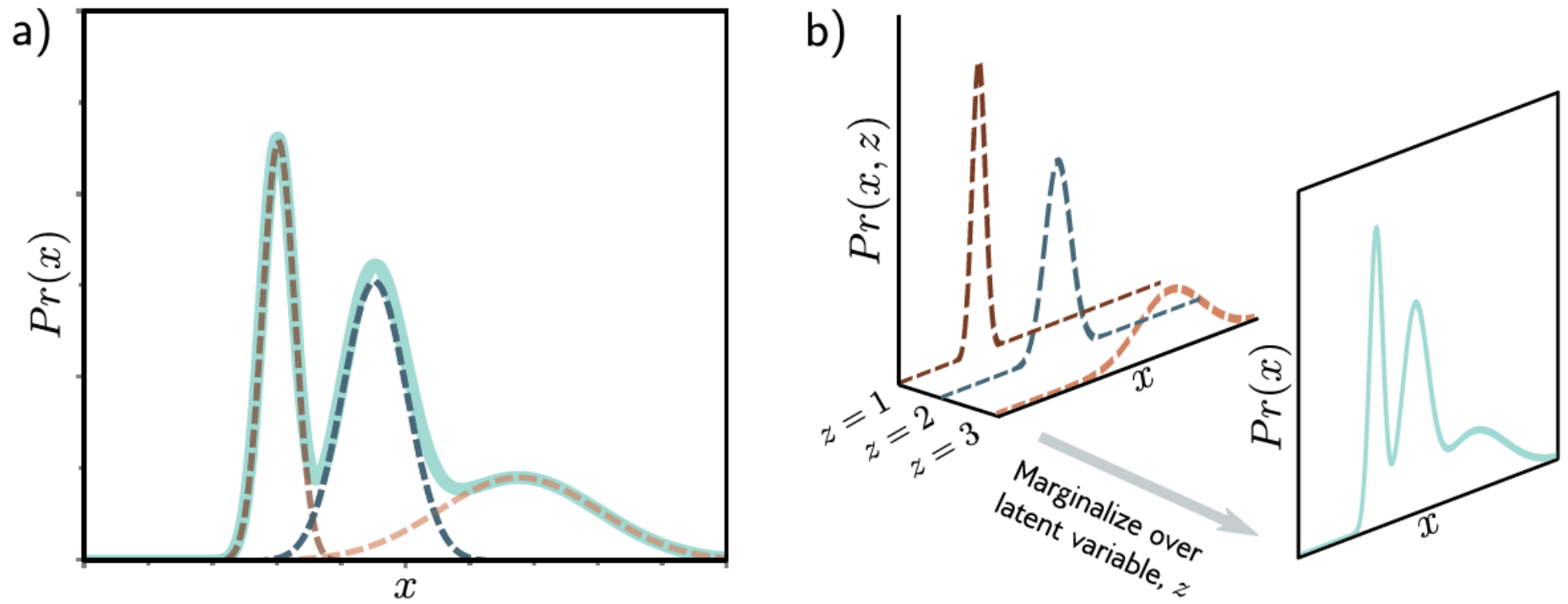


Figure 17.1 Mixture of Gaussians (MoG). a) The MoG describes a complex probability distribution (cyan curve) as a weighted sum of Gaussian components (dashed curves). b) This sum is the marginalization of the joint density $Pr(x, z)$ between the continuous observed data x and a discrete latent variable z.

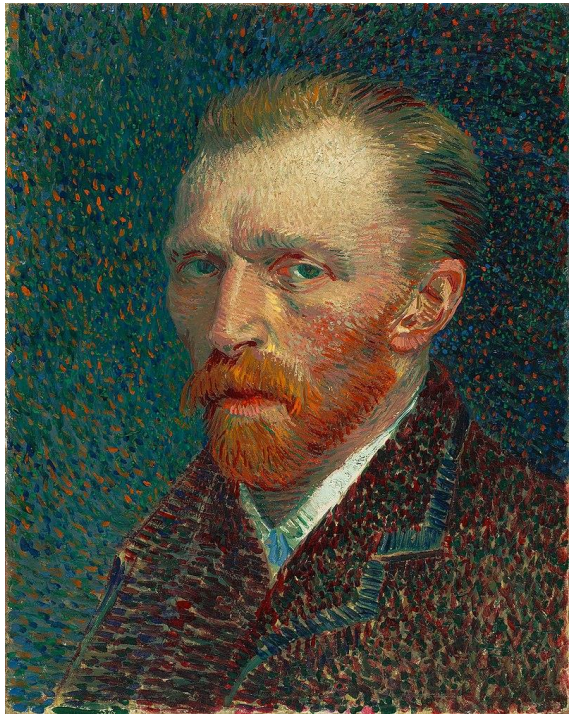
Nonlinear latent variable model [1]

- Both ***x*** and ***z*** are continuous and multivariate
- Start with a prior *Pr*(***z***) which is normally distributed
- Our deep neural network with parameters ϕ calculates f[***z***, ϕ]
- Use the likelihood, which now also depends on neural network parameters ϕ, written *Pr*(***x***|***z***, ϕ), also normally distributed, with mean from our neural network and fixed spherical covariance $\sigma^2\mathbf{I}$

$$\begin{aligned} Pr(\mathbf{x}|\boldsymbol{\phi}) &= \int Pr(\mathbf{x},\mathbf{z}|\boldsymbol{\phi})d\mathbf{z} \\ &= \int Pr(\mathbf{x}|\mathbf{z},\boldsymbol{\phi}) \cdot Pr(\mathbf{z})d\mathbf{z} \\ &= \int \mathrm{Norm}_{\mathbf{x}}\Big[\mathbf{f}[\mathbf{z},\boldsymbol{\phi}],\sigma^2\mathbf{I}\Big] \cdot \mathrm{Norm}_{\mathbf{z}}\left[\mathbf{0},\mathbf{I}\right] d\mathbf{z}. \end{aligned} \tag{17.7}$$

Nonlinear latent variable model [2]

- Our 1D mixture of Gaussians was a sum of a small number of 1D Gaussian distributions
- The author says, for our nonlinear multivariate latent model:
  "This can be viewed as an infinite weighted sum (i.e., an infinite mixture) of spherical Gaussians with different means, where the weights are $Pr(\boldsymbol{z})$ and the means are the network outputs $f[\boldsymbol{z}, \phi]$"
- Artists like Vincent van Gogh using the pointillism technique can make an entire painting from dots. This isn't exactly an analogous situation, but the idea is similar that you can build up probability density from many small spherical Gaussians



Generation for nonlinear latent variable model

- Sample a $\boldsymbol{z}^*$, which is easy because $Pr(\boldsymbol{z})$ is Gaussian
- Run $\boldsymbol{z}^*$ through neural network and use output $\mathbf{f}[\boldsymbol{z}^*, \phi]$ as mean to sample $\boldsymbol{x}^*$, which is also easy because likelihood $Pr(\boldsymbol{x}|\boldsymbol{z}, \phi)$ is also Gaussian

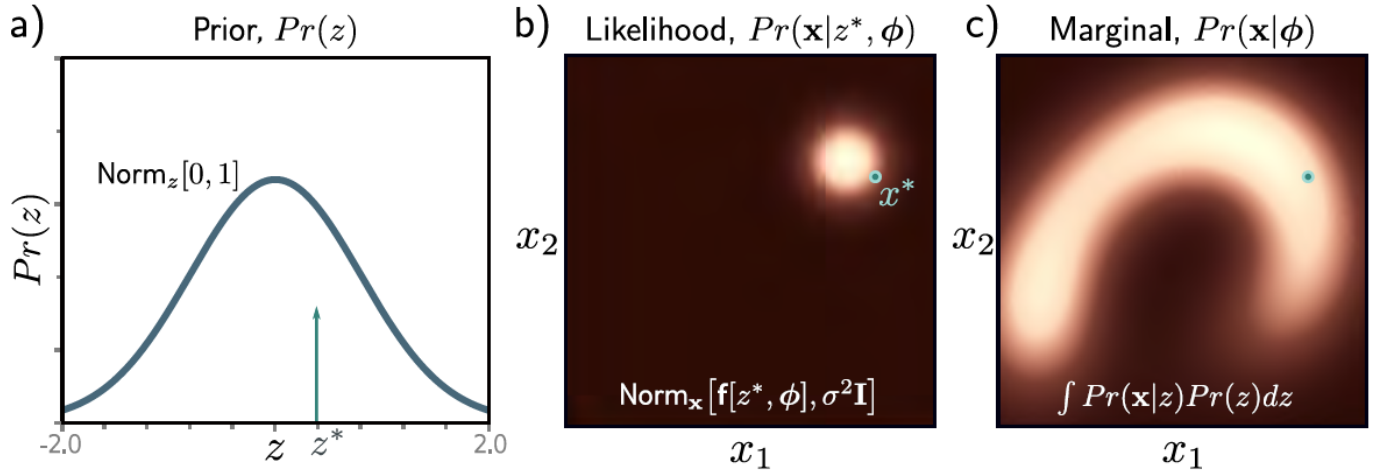


Figure 17.3 Generation from nonlinear latent variable model. a) We draw a sample z^* from the prior probability $Pr(z)$ over the latent variable. b) A sample $\mathbf{x}^*$ is then drawn from $Pr(\mathbf{x}|z^*, \phi)$. This is a spherical Gaussian with a mean that is a nonlinear function $\mathbf{f}[\bullet, \phi]$ of z^* and a fixed variance $\sigma^2\mathbf{I}$. c) If we repeat this process many times, we recover the density $Pr(\mathbf{x}|\phi)$.

Training the nonlinear latent variable model

- We'd like to train by picking parameters ϕ that maximize the log likelihood over the training dataset
- Recall our fully expanded probability is:

$$Pr(\mathbf{x}_i|\boldsymbol{\phi}) = \int \text{Norm}_{\mathbf{x}_i}[\mathbf{f}[\mathbf{z},\boldsymbol{\phi}],\sigma^2\mathbf{I}] \cdot \text{Norm}_{\mathbf{z}}[\mathbf{0},\mathbf{I}]d\mathbf{z}. \quad (17.9)$$

- But unfortunately, this is intractable
- If we can find a lower bound for the log-likelihood that is very close to the actual log-likelihood, that's probably good enough
- We will create an Evidence Lower Bound (ELBO), but first we will need Jensen's inequality

Jensen's inequality [1]

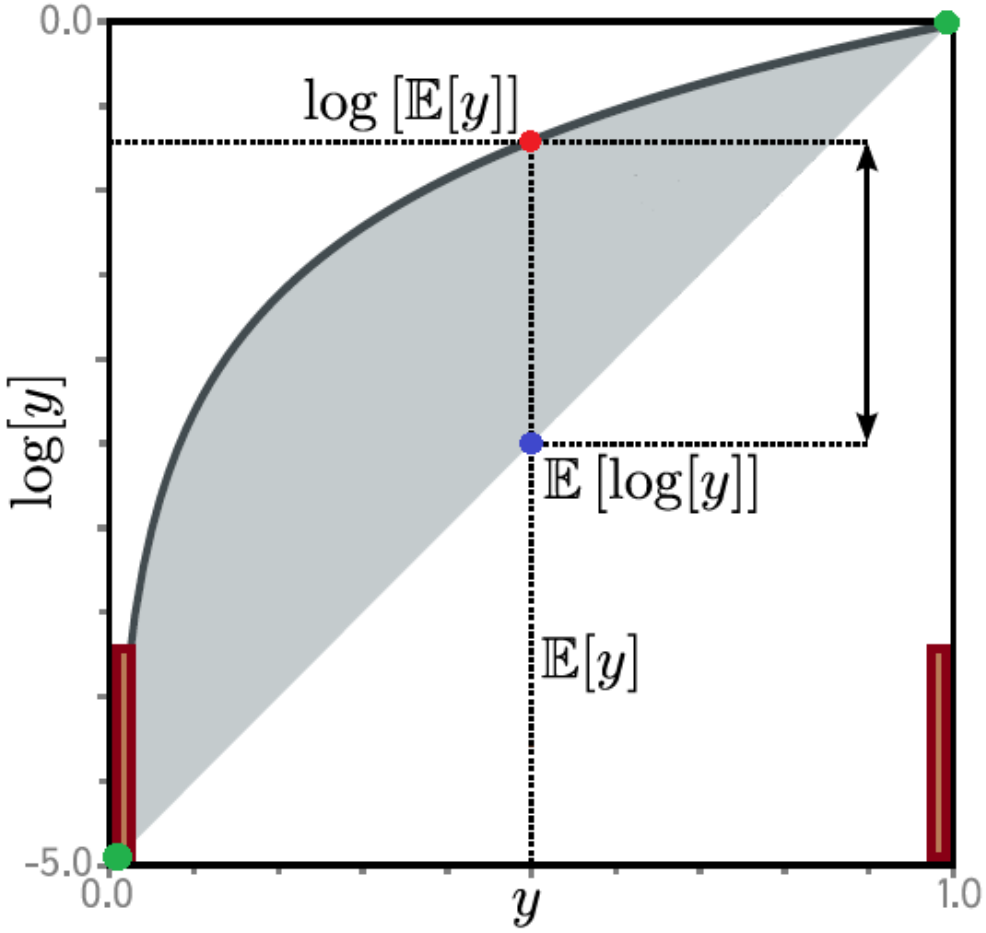
- If we have a concave function (curved downward, second derivative is always negative), the output of the function on the single-valued expectation of some data inputs is always greater than or equal to the expectation of the function of the data.
- For data *y* and concave function *g*(): $g[\mathrm{E}[y]] >= \mathrm{E}[g[y]]$
- Logarithm is a concave function, and expanding expectation, we get:

$$\log\left[\int Pr(y)ydy\right] \geq \int Pr(y)\log[y]dy. \qquad (17.12)$$

- More generally, we can replace *y* with *h*[*y*] and the inequality holds:

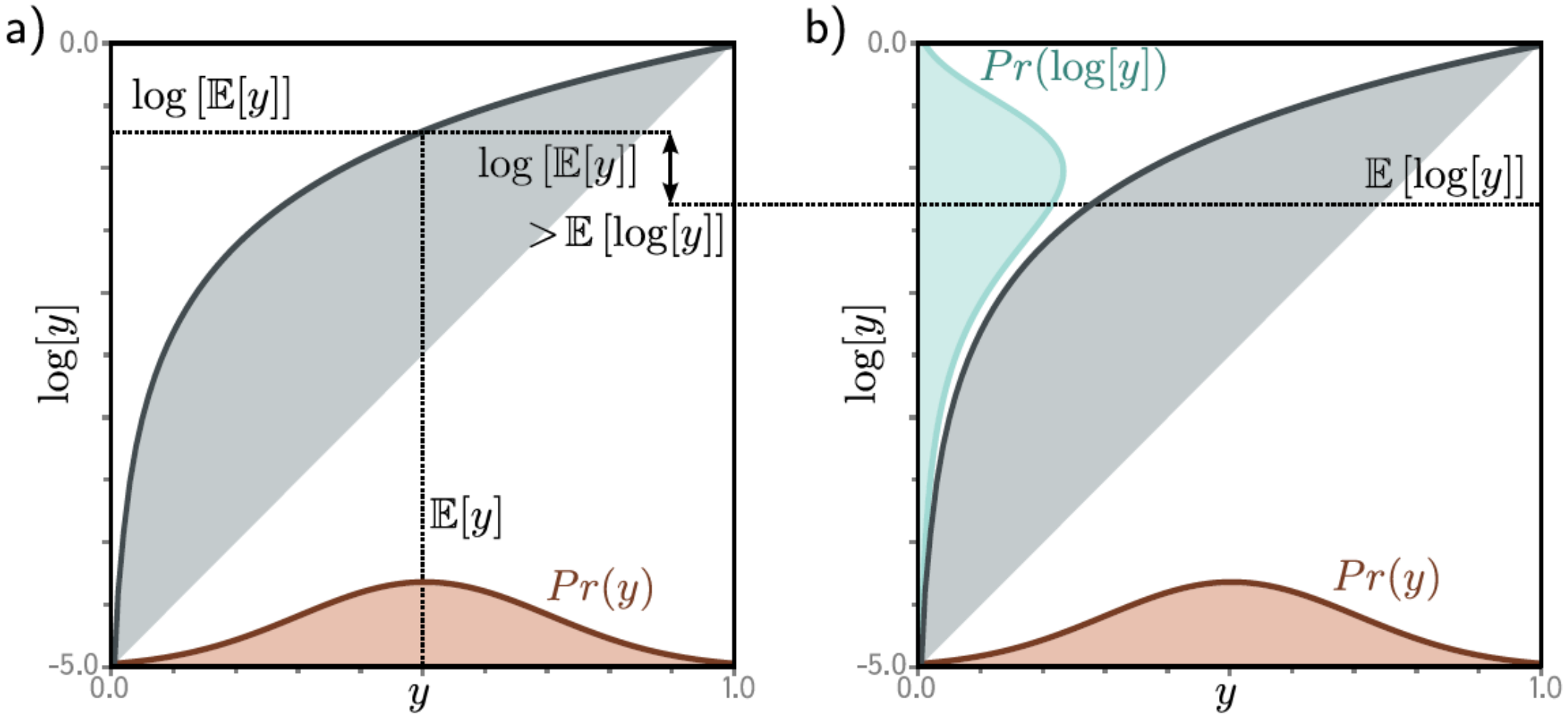
$$\log\left[\int Pr(y)h[y]dy\right] \geq \int Pr(y)\log[h[y]]dy. \qquad (17.13)$$

Jensen's inequality [2]



- A simplified example of our data is two point masses, each with half the probability
- $\mathrm{E}[y]$ is just the midpoint/average
- $\log\mathrm{E}[y]$ is the red point on the curve
- $\mathrm{E}[\log[y]]$ is the midpoint/average of the two green points for $\log[y]$ and is the blue point in the center
- More generally, $\mathrm{E}[\log[y]]$ will always be in the gray shaded area, but $\log\mathrm{E}[y]$ will be on the curve above

Jensen's inequality [3]



ELBO [1]

- We take our log-likelihood, and multiply and divide it by a new probability distribution $q(\mathbf{z})$, giving us the log of an expectation of $q(\mathbf{z})$:

$$\log\left[\int q(\mathbf{z})\frac{Pr(\mathbf{x},\mathbf{z}|\phi)}{q(\mathbf{z})}d\mathbf{z}\right], \qquad (17.14)$$

- Using Jensen's inequality, the log of the expectation is greater than or equal to the expectation of the log:

$$\log\left[\int q(\mathbf{z})\frac{Pr(\mathbf{x},\mathbf{z}|\phi)}{q(\mathbf{z})}d\mathbf{z}\right] \geq \int q(\mathbf{z})\log\left[\frac{Pr(\mathbf{x},\mathbf{z}|\phi)}{q(\mathbf{z})}\right]d\mathbf{z}, \qquad (17.15)$$

- The right side above, in the box, is the *evidence lower bound* (ELBO)
- If $q(\mathbf{z})$ is parameterized by θ, we write:

$$\text{ELBO}[\boldsymbol{\theta},\boldsymbol{\phi}] = \int q(\mathbf{z}|\boldsymbol{\theta})\log\left[\frac{Pr(\mathbf{x},\mathbf{z}|\boldsymbol{\phi})}{q(\mathbf{z}|\boldsymbol{\theta})}\right]d\mathbf{z}. \qquad (17.16)$$

ELBO [2]

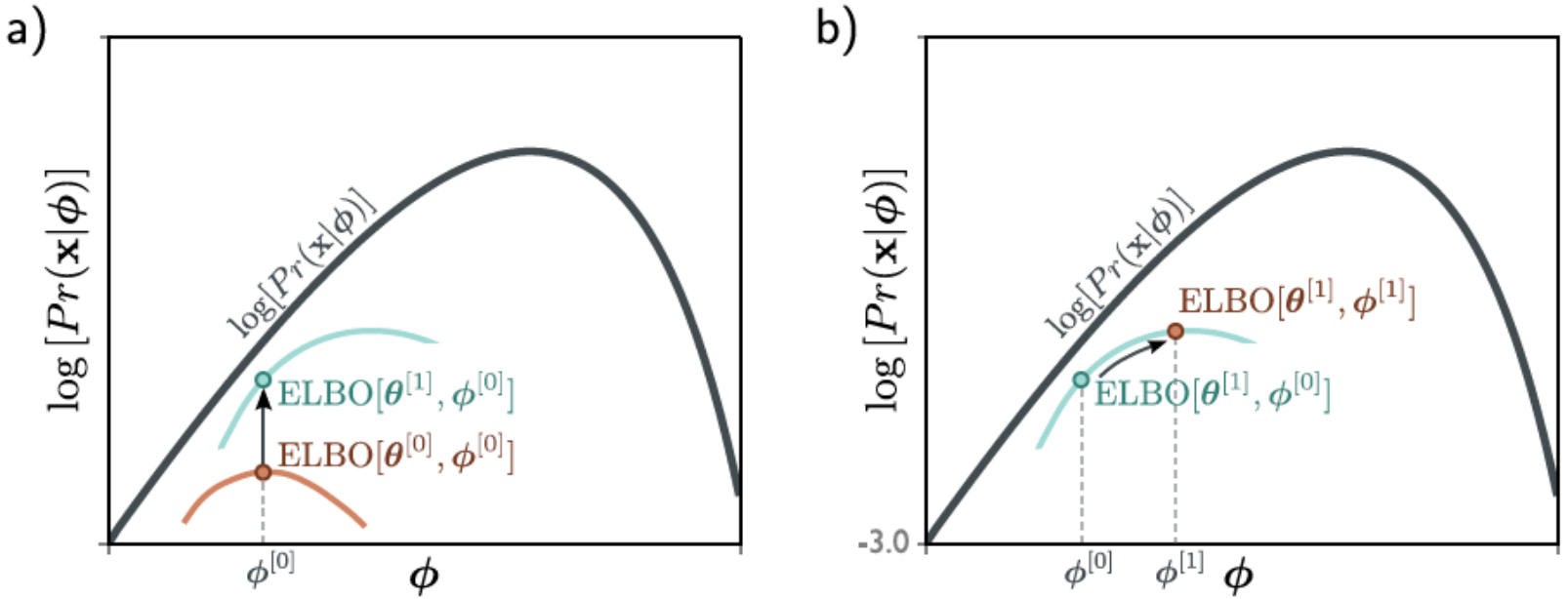


Figure 17.6 Evidence lower bound (ELBO). The goal is to maximize the log-likelihood $\log[Pr(\mathbf{x}|\boldsymbol{\phi})]$ (black curve) with respect to the parameters $\boldsymbol{\phi}$. The ELBO is a function that lies everywhere below the log-likelihood. It is a function of both $\boldsymbol{\phi}$ and a second set of parameters $\boldsymbol{\theta}$. For fixed $\boldsymbol{\theta}$, we get a function of $\boldsymbol{\phi}$ (two colored curves for different values of $\boldsymbol{\theta}$). Consequently, we can increase the log-likelihood by either improving the ELBO with respect to a) the new parameters $\boldsymbol{\theta}$ (moving from colored curve to colored curve) or b) the original parameters $\boldsymbol{\phi}$ (moving along the current colored curve).

ELBO [3]

- The book shows in section 17.4.1, we can reformulate the ELBO:

$$\text{ELBO}[\boldsymbol{\theta}, \boldsymbol{\phi}] = \log\left[Pr(\mathbf{x}|\boldsymbol{\phi})\right] - \text{D}_{KL}\left[q(\mathbf{z}|\boldsymbol{\theta})\middle\|Pr(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})\right]. \quad (17.17)$$

- The above shows the ELBO differs by the KL divergence between the $q(\boldsymbol{z},\theta)$ we choose and the conditional probability $Pr(\boldsymbol{z}|\boldsymbol{x}, \phi)$

- Another reformulation in section 17.4.2 is:

$$\text{ELBO}[\boldsymbol{\theta}, \boldsymbol{\phi}] = \int q(\mathbf{z}|\boldsymbol{\theta}) \log\left[Pr(\mathbf{x}|\mathbf{z}, \boldsymbol{\phi})\right] d\mathbf{z} - \text{D}_{KL}\left[q(\mathbf{z}|\boldsymbol{\theta})\middle\|Pr(\mathbf{z})\right], \quad (17.18)$$

- Here, the first term measures the average agreement (likelihood) of our latent variable and the training data, and is called the *reconstruction loss*. This form is amenable to training a neural network.

Variational approximation

- The ELBO is tight when $q(\boldsymbol{z})$ matches the posterior $Pr(\boldsymbol{z}|\boldsymbol{x}, \phi)$
- We could use Bayes' rule to calculate this posterior, but as with most real world problems, the denominator is intractable
- Instead, we choose a variational approximation, parameterizing as $q(\boldsymbol{z}, \theta)$ and seeking the best parameters θ
- It is common to choose $q(\boldsymbol{z}, \theta)$ to be multivariate normal
- The posterior depends on the data **x**, and we can do the same for $q(\boldsymbol{z})$, turning it into:

$$q(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta}) = \mathrm{Norm}_{\mathbf{z}}\Big[\mathbf{g}_{\boldsymbol{\mu}}[\mathbf{x}, \boldsymbol{\theta}], \mathbf{g}_{\boldsymbol{\Sigma}}[\mathbf{x}, \boldsymbol{\theta}]\Big], \tag{17.20}$$

Variational autoencoder [1]

- We want to compute our latest form of the ELBO:

$$\text{ELBO}[\boldsymbol{\theta}, \boldsymbol{\phi}] = \int q(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta}) \log\big[Pr(\mathbf{x}|\mathbf{z}, \boldsymbol{\phi})\big] d\mathbf{z} - \mathrm{D}_{KL}\Big[q(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})\Big|\Big|Pr(\mathbf{z})\Big], \qquad (17.21)$$

- The first term is an intractable integral, but we can approximate any expectation by sampling, here using just one data point ***x***:

$$\text{ELBO}[\boldsymbol{\theta}, \boldsymbol{\phi}] \approx \log\big[Pr(\mathbf{x}|\mathbf{z}^*, \boldsymbol{\phi})\big] - \mathrm{D}_{KL}\Big[q(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})\Big|\Big|Pr(\mathbf{z})\Big]. \qquad (17.23)$$

- The second term is the KL divergence, and if we choose two normal distributions, it has a closed form equation
- Combining the above, we can approximate the ELBO for any given training data point ***x***

Variational autoencoder [2]

- This all started with trying to maximize the likelihood of our data, so we will use the negative of the ELBO as our loss function (to minimize)
- To compute the ELBO for a data point $\boldsymbol{x}$:
  - Use our second neural network to output $g[\boldsymbol{x}, \theta]$, predicting mean μ and covariance Σ
  - Draw a random sample from this normal distribution, approximating $q(\boldsymbol{z}|\boldsymbol{x}, \theta)$
  - Compute the ELBO using equation 17.23 from the previous slide, which requires our first neural network to calculate $f[\boldsymbol{z}^*, \phi]$, predicting our likelihood $Pr(\boldsymbol{x}|\boldsymbol{z}, \phi)$
- This is probably easier to see in a diagram, with the encoder and decoder labeled

Variational autoencoder [3]

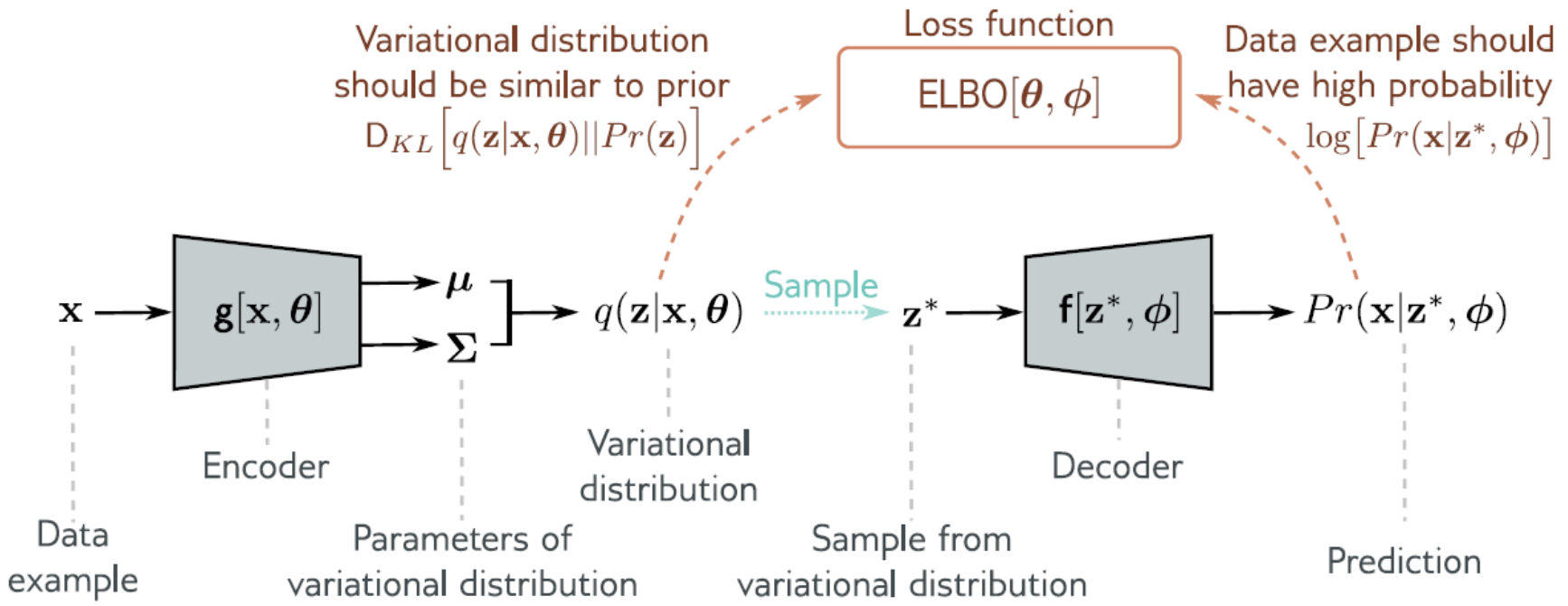
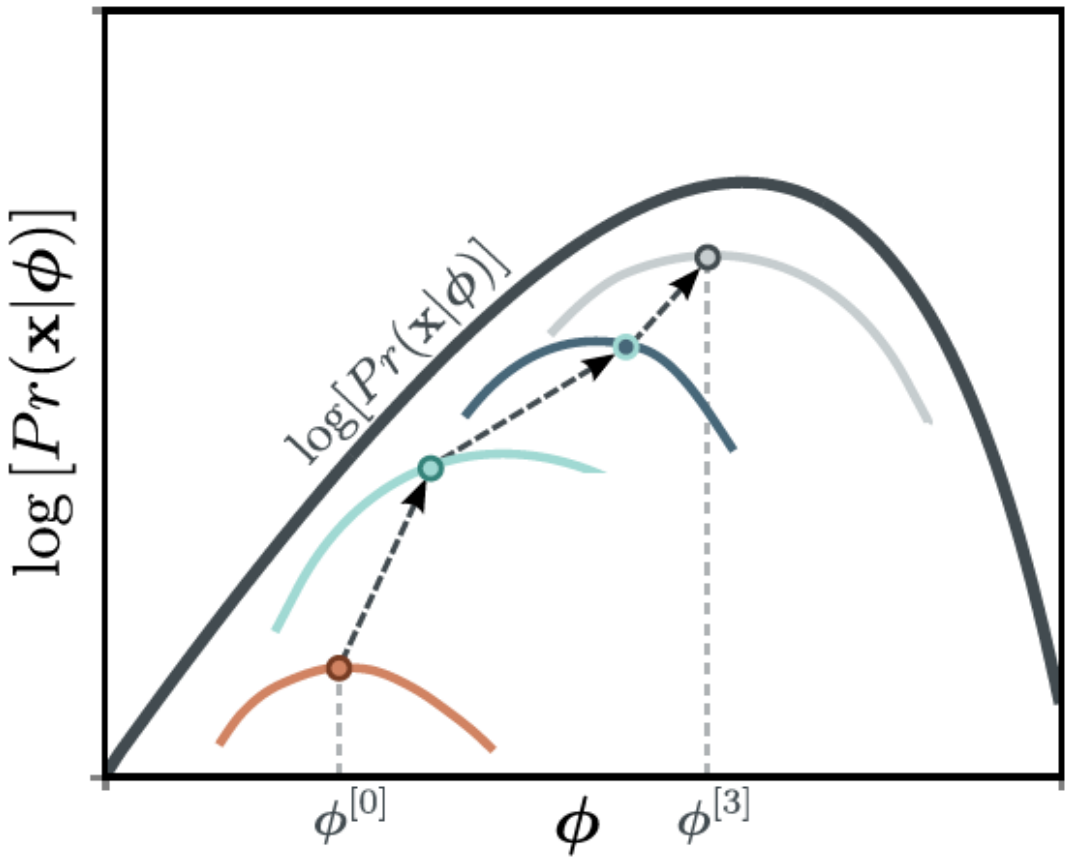


Figure 17.9 Variational autoencoder. The encoder $\mathbf{g}[\mathbf{x},\boldsymbol{\theta}]$ takes a training example $\mathbf{x}$ and predicts the parameters $\boldsymbol{\mu},\boldsymbol{\Sigma}$ of the variational distribution $q(\mathbf{z}|\mathbf{x},\boldsymbol{\theta})$. We sample from this distribution and then use the decoder $\mathbf{f}[\mathbf{z},\boldsymbol{\phi}]$ to predict the data $\mathbf{x}$. The loss function is the negative ELBO, which depends on how accurate this prediction is and how similar the variational distribution $q(\mathbf{z}|\mathbf{x},\boldsymbol{\theta})$ is to the prior $Pr(\mathbf{z})$ (equation 17.21).

Variational autoencoder [4]

Figure 17.10 The VAE updates both factors that determine the lower bound at each iteration. Both the parameters ϕ of the decoder and the parameters θ of the encoder are manipulated to increase this lower bound.



The reparameterization trick

- We have one last problem with our formulation: the middle step involved drawing a random sample from our normal distribution
- You can't run backpropagation for sampling because there are no gradients
- Fortunately a simple reparameterization trick takes advantage of sampling $\boldsymbol{z}^*$ from a normal with mean μ and covariance Σ is the same as sampling ϵ^* from Norm[0,$\mathbf{I}$] and translating it:

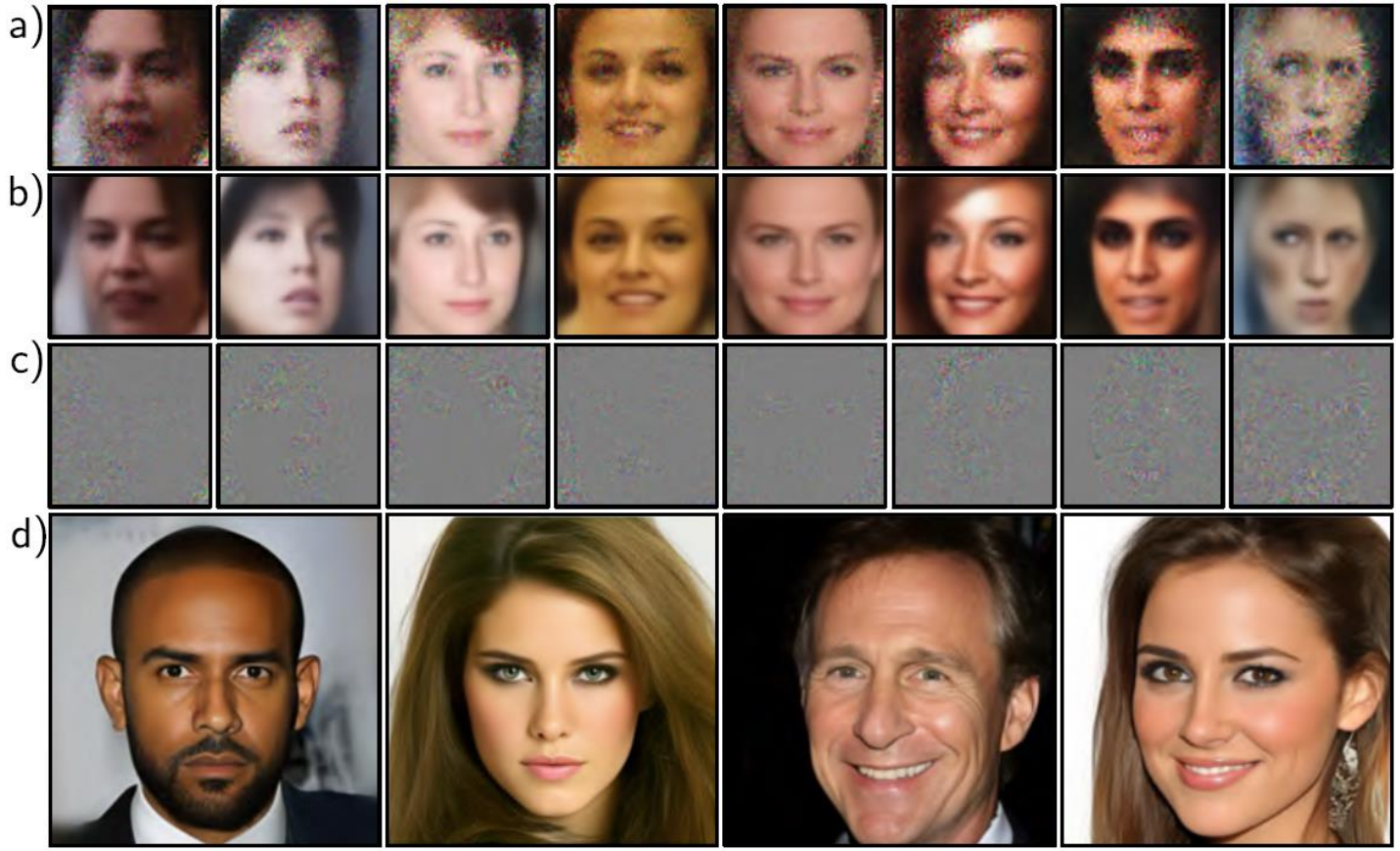
$$\mathbf{z}^* = \boldsymbol{\mu} + \boldsymbol{\Sigma}^{1/2}\boldsymbol{\epsilon}^*, \qquad (17.25)$$

VAE applications [1]

- *Approximating sample probability*
- You can’t directly approximate probability with a VAE
- A technique called *importance sampling* weights/rescales the samples such that you may be able to approximate

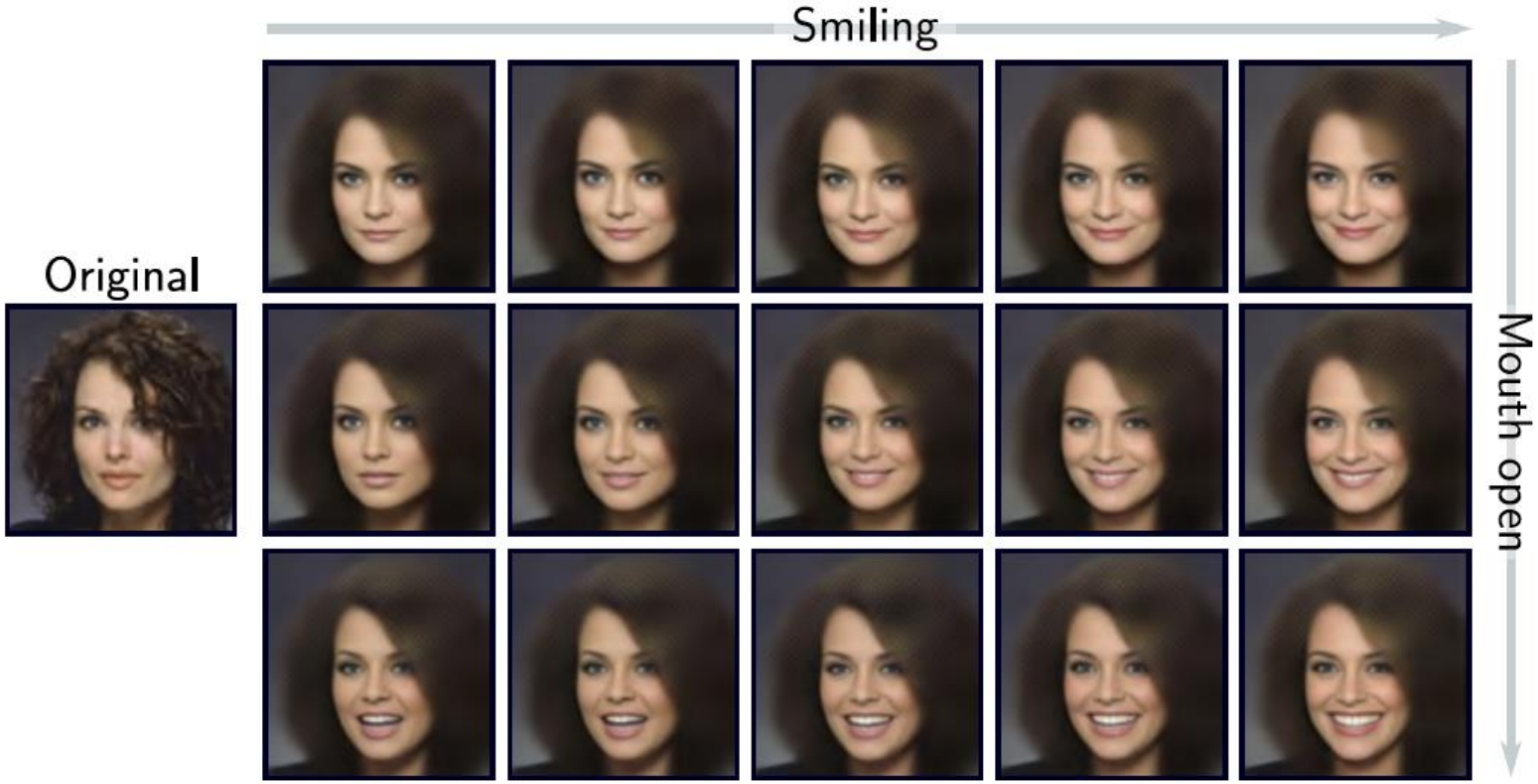
VAE applications [2]

- *Generation*
- Many assumptions we've discussed lead to noise. For images, that means you get poor quality and blurry images.
- More complicated architectures, such as hierarchical priors improve quality



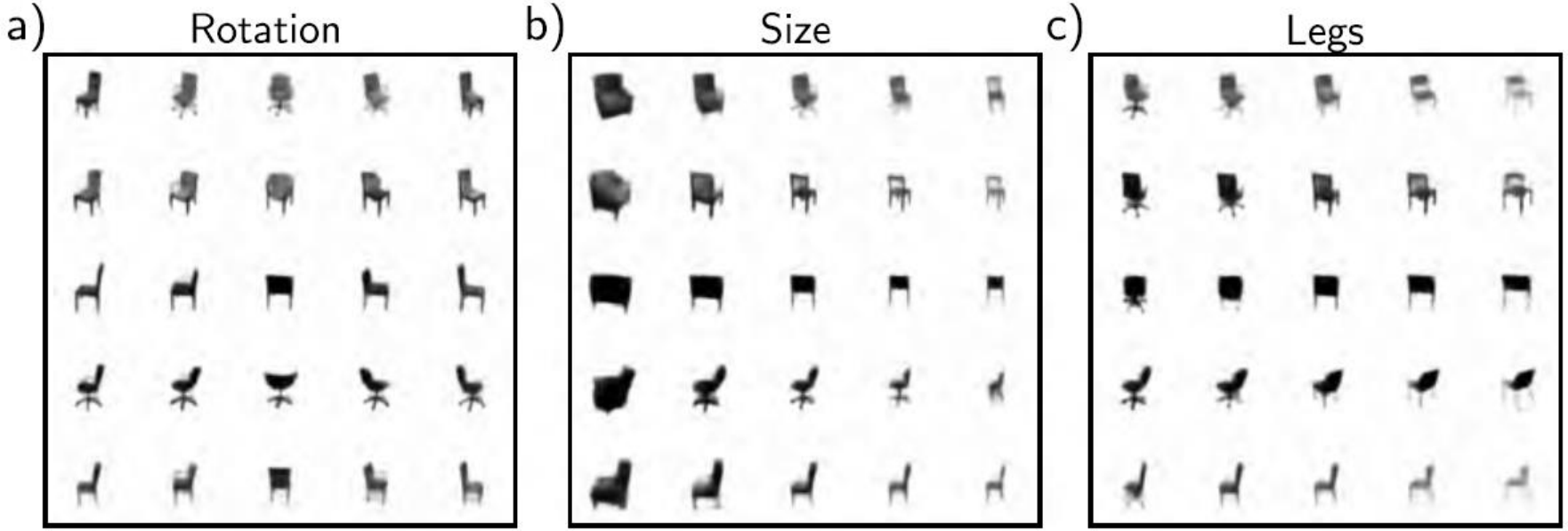
VAE applications [3]

- *Resynthesis*
- You can modify data, such as images. Find the difference between two concepts in latent space (such as smiling vs. not smiling), then add that difference to a specific example.



VAE applications [4]

- *Disentanglement*
- You'd like to have human-interpretable features be represented by separate directions in latent space
- The Beta VAE model upweights the KL divergence term
- The Total correlation VAE adds a loss to decrease total correlation



VAE summary

- The VAE allows us to learn a complicated latent variable model
- We generate new examples by running the encoder neural network, sampling from the latent variable, passing the sample through the decoder neural network, then adding independent Gaussian noise
- This differs from a simple autoencoder because the middle of an autoencoder is a single, deterministic embedding, but the middle of a VAE is a probability distribution
- Maximum likelihood training of a VAE requires estimating a lower bound on the likelihood, using a variational approximation, and estimating using samples from our training data