Recap of last lecture

1. Autoregressive models:
   - Chain rule based factorization is fully general
   - Compact representation via *conditional independence* and/or *neural parameterizations*
2. Autoregressive models Pros:
   - Easy to evaluate likelihoods
   - Easy to train
3. Autoregressive models Cons:
   - Requires an ordering
   - Generation is sequential
   - Cannot learn features in an unsupervised way

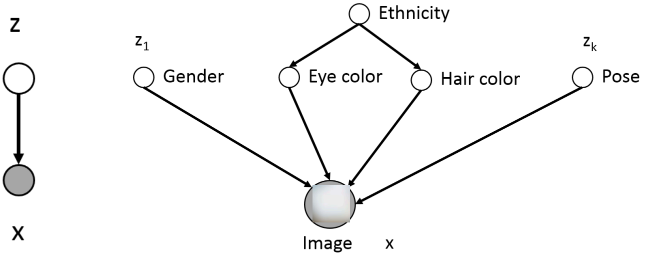
Plan for today

1. Latent Variable Models
    - Mixture models
    - Variational autoencoder
    - Variational inference and learning

Latent Variable Models: Motivation

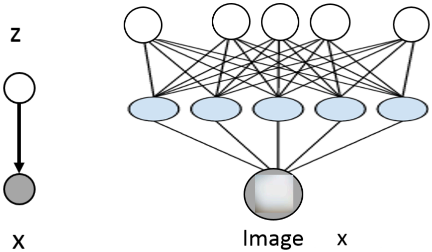
1. Lots of variability in images $\mathbf{x}$ due to gender, eye color, hair color, pose, etc. However, unless images are annotated, these factors of variation are not explicitly available (latent).
2. **Idea**: explicitly model these factors using latent variables $\mathbf{z}$

Latent Variable Models: Motivation



1. Only shaded variables $\mathbf{x}$ are observed in the data (pixel values)
2. Latent variables $\mathbf{z}$ correspond to high level features
   - If $\mathbf{z}$ chosen properly, $p(\mathbf{x}|\mathbf{z})$ could be much simpler than $p(\mathbf{x})$
   - If we had trained this model, then we could identify features via $p(\mathbf{z} \mid \mathbf{x})$, e.g., $p(EyeColor = Blue|\mathbf{x})$
3. **Challenge:** Very difficult to specify these conditionals by hand

Deep Latent Variable Models

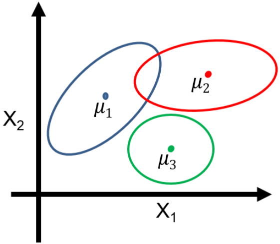


- Use neural networks to model the conditionals (deep latent variable models):
  1. $\mathbf{z} \sim \mathcal{N}(0, I)$
  2. $p(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}\left(\mu_\theta(\mathbf{z}), \Sigma_\theta(\mathbf{z})\right)$ where μ_θ,Σ_θ are neural networks
- *Hope* that after training, **z** will correspond to meaningful latent factors of variation (*features*). Unsupervised representation learning.
- As before, features can be computed via $p(\mathbf{z} \mid \mathbf{x})$

Mixture of Gaussians: a Shallow Latent Variable Model

Mixture of Gaussians. Bayes net: $\mathbf{z} \rightarrow \mathbf{x}$.

1. $\mathbf{z} \sim \text{Categorical}(1, \cdots, K)$
2. $p(\mathbf{x} \mid \mathbf{z} = k) = \mathcal{N}(\mu_k, \Sigma_k)$



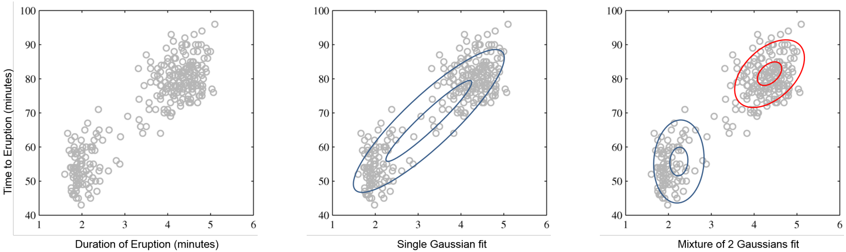
Generative process

1. Pick a mixture component k by sampling z
2. Generate a data point by sampling from that Gaussian

Mixture of Gaussians: a Shallow Latent Variable Model

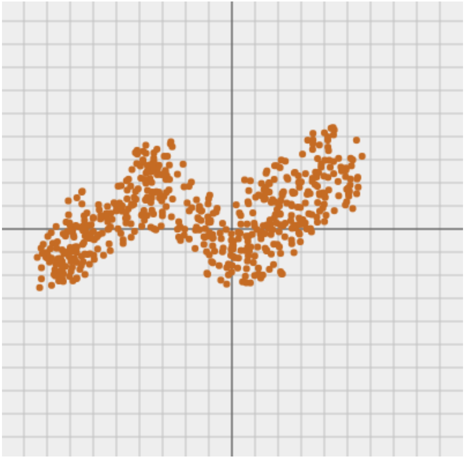
Mixture of Gaussians:

1. $\mathbf{z} \sim \mathrm{Categorical}(1, \cdots, K)$
2. $p(\mathbf{x} \mid \mathbf{z} = k) = \mathcal{N}(\mu_k, \Sigma_k)$

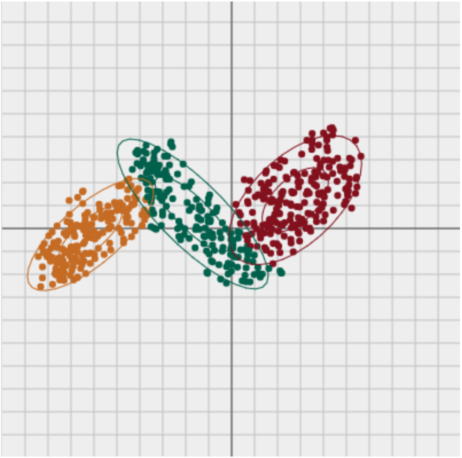


- **Clustering:** The posterior $p(\mathbf{z} \mid \mathbf{x})$ identifies the mixture component
- **Unsupervised learning:** We are hoping to learn from unlabeled data (ill-posed problem)

Unsupervised learning

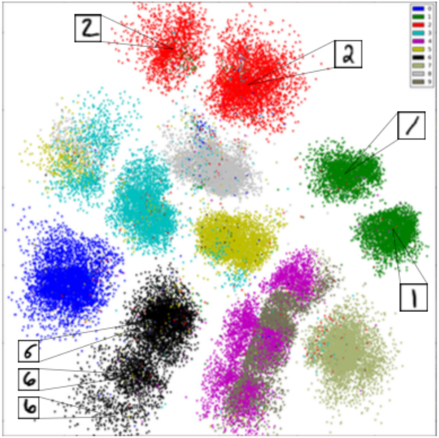


Unsupervised learning



Shown is the posterior probability that a data point was generated by the i-th mixture component, $P(z = i|x)$

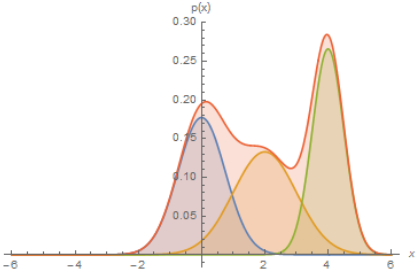
Unsupervised learning



Unsupervised clustering of handwritten digits.

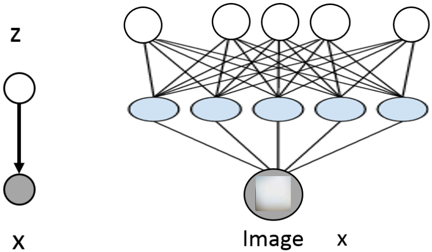
Mixture models

Alternative motivation: Combine simple models into a more complex and expressive one



$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) = \sum_{\mathbf{z}} p(\mathbf{z}) p(\mathbf{x} \mid \mathbf{z}) = \sum_{k=1}^{K} p(\mathbf{z} = k) \underbrace{\mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)}_{\text{component}}$$

Variational Autoencoder



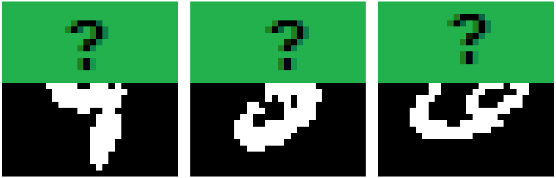
A mixture of an infinite number of Gaussians:

1. $\mathbf{z} \sim \mathcal{N}(0, I)$
2. $p(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mu_\theta(\mathbf{z}), \Sigma_\theta(\mathbf{z}))$ where μ_θ,Σ_θ are neural networks
   - $\mu_\theta(\mathbf{z}) = \sigma(A\mathbf{z} + c) = (\sigma(a_1\mathbf{z} + c_1), \sigma(a_2\mathbf{z} + c_2)) = (\mu_1(\mathbf{z}), \mu_2(\mathbf{z}))$
   - $\Sigma_\theta(\mathbf{z}) = diag(\exp(\sigma(B\mathbf{z} + d))) = \begin{pmatrix} \exp(\sigma(b_1\mathbf{z}+d_1)) & 0 \\ 0 & \exp(\sigma(b_2\mathbf{z}+d_2)) \end{pmatrix}$
   - $\theta = (A, B, c, d)$
3. Even though $p(\mathbf{x} \mid \mathbf{z})$ is simple, the marginal $p(\mathbf{x})$ is very complex/flexible

Recap

- Latent Variable Models
  - Allow us to define complex models $p(\mathbf{x})$ in terms of simpler building blocks $p(\mathbf{x} \mid \mathbf{z})$
  - Natural for unsupervised learning tasks (clustering, unsupervised representation learning, etc.)
  - No free lunch: much more difficult to learn compared to fully observed, autoregressive models

Marginal Likelihood



- Suppose some pixel values are missing at train time (e.g., top half)
- Let $\mathbf{X}$ denote observed random variables, and $\mathbf{Z}$ the unobserved ones (also called hidden or latent)
- Suppose we have a model for the joint distribution (e.g., PixelCNN)

$$p(\mathbf{X}, \mathbf{Z}; \theta)$$

What is the probability $p(\mathbf{X} = \bar{\mathbf{x}}; \theta)$ of observing a training data point $\bar{\mathbf{x}}$?

$$\sum_{\mathbf{z}} p(\mathbf{X} = \bar{\mathbf{x}}, \mathbf{Z} = \mathbf{z}; \theta) = \sum_{\mathbf{z}} p(\bar{\mathbf{x}}, \mathbf{z}; \theta)$$

- Need to consider all possible ways to complete the image (fill green part)

Variational Autoencoder Marginal Likelihood

A mixture of an infinite number of Gaussians:

1. $\mathbf{z} \sim \mathcal{N}(0, I)$
2. $p(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}\left(\mu_\theta(\mathbf{z}), \Sigma_\theta(\mathbf{z})\right)$ where μ_θ,Σ_θ are neural networks
3. $\mathbf{Z}$ are unobserved at train time (also called hidden or latent)
4. Suppose we have a model for the joint distribution. What is the probability $p(\mathbf{X} = \bar{\mathbf{x}}; \theta)$ of observing a training data point $\bar{\mathbf{x}}$?

$$\int_{\mathbf{z}} p(\mathbf{X} = \bar{\mathbf{x}}, \mathbf{Z} = \mathbf{z}; \theta) d\mathbf{z} = \int_{\mathbf{z}} p(\bar{\mathbf{x}}, \mathbf{z}; \theta) d\mathbf{z}$$

Partially observed data

- Suppose that our joint distribution is
$$p(\mathbf{X}, \mathbf{Z}; \theta)$$
- We have a dataset $\mathcal{D}$, where for each datapoint the $\mathbf{X}$ variables are observed (e.g., pixel values) and the variables $\mathbf{Z}$ are never observed (e.g., cluster or class id.). $\mathcal{D} = \{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(M)}\}$.
- Maximum likelihood learning:
$$\log \prod_{\mathbf{x} \in \mathcal{D}} p(\mathbf{x}; \theta) = \sum_{\mathbf{x} \in \mathcal{D}} \log p(\mathbf{x}; \theta) = \sum_{\mathbf{x} \in \mathcal{D}} \log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}; \theta)$$
- Evaluating $\log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}; \theta)$ can be intractable. Suppose we have 30 binary latent features, $\mathbf{z} \in \{0, 1\}^{30}$. Evaluating $\sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}; \theta)$ involves a sum with 2^{30} terms. For continuous variables, $\log \int_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}; \theta) d\mathbf{z}$ is often intractable. Gradients ∇_θ also hard to compute.
- Need **approximations**. One gradient evaluation per training data point $\mathbf{x} \in \mathcal{D}$, so approximation needs to be cheap.

First attempt: Naive Monte Carlo

Likelihood function $p_\theta(\mathbf{x})$ for Partially Observed Data is hard to compute:

$$p_\theta(\mathbf{x}) = \sum_{\text{All values of } \mathbf{z}} p_\theta(\mathbf{x}, \mathbf{z}) = |\mathcal{Z}| \sum_{\mathbf{z} \in \mathcal{Z}} \frac{1}{|\mathcal{Z}|} p_\theta(\mathbf{x}, \mathbf{z}) = |\mathcal{Z}| \mathbb{E}_{\mathbf{z} \sim Uniform(\mathcal{Z})} \left[p_\theta(\mathbf{x}, \mathbf{z}) \right]$$

We can think of it as an (intractable) expectation. Monte Carlo to the rescue:

1. Sample $\mathbf{z}^{(1)}, \cdots, \mathbf{z}^{(k)}$ uniformly at random
2. Approximate expectation with sample average

$$\sum_{\mathbf{z}} p_\theta(\mathbf{x}, \mathbf{z}) \approx |\mathcal{Z}| \frac{1}{k} \sum_{j=1}^{k} p_\theta(\mathbf{x}, \mathbf{z}^{(j)})$$

Works in theory but not in practice. For most $\mathbf{z}$, $p_\theta(\mathbf{x}, \mathbf{z})$ is very low (most completions don't make sense). Some completions have large $p_\theta(\mathbf{x}, \mathbf{z})$ but we will never "hit" likely completions by uniform random sampling. Need a clever way to select $\mathbf{z}^{(j)}$ to reduce variance of the estimator.

Second attempt: Importance Sampling

Likelihood function $p_\theta(\mathbf{x})$ for Partially Observed Data is hard to compute:

$$p_\theta(\mathbf{x}) = \sum_{\text{All possible values of } \mathbf{z}} p_\theta(\mathbf{x}, \mathbf{z}) = \sum_{\mathbf{z} \in \mathcal{Z}} \frac{q(\mathbf{z})}{q(\mathbf{z})} p_\theta(\mathbf{x}, \mathbf{z}) = \mathbb{E}_{\mathbf{z} \sim q(\mathbf{z})} \left[\frac{p_\theta(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \right]$$

Monte Carlo to the rescue:

1. Sample $\mathbf{z}^{(1)}, \cdots, \mathbf{z}^{(k)}$ from $q(\mathbf{z})$
2. Approximate expectation with sample average

$$p_\theta(\mathbf{x}) \approx \frac{1}{k} \sum_{j=1}^{k} \frac{p_\theta(\mathbf{x}, \mathbf{z}^{(j)})}{q(\mathbf{z}^{(j)})}$$

   What is a good choice for $q(\mathbf{z})$? Intuitively, frequently sample $\mathbf{z}$ (completions) that are likely given $\mathbf{x}$ under $p_\theta(\mathbf{x}, \mathbf{z})$.

3. This is an unbiased estimator of $p_\theta(\mathbf{x})$

$$\mathbb{E}_{\mathbf{z}^{(j)}) \sim q(\mathbf{z})} \left[\frac{1}{k} \sum_{j=1}^{k} \frac{p_\theta(\mathbf{x}, \mathbf{z}^{(j)})}{q(\mathbf{z}^{(j)})} \right] = p_\theta(\mathbf{x})$$

Estimating log-likelihoods

Likelihood function $p_\theta(\mathbf{x})$ for Partially Observed Data is hard to compute:

$$p_\theta(\mathbf{x}) = \sum_{\text{All possible values of } \mathbf{z}} p_\theta(\mathbf{x}, \mathbf{z}) = \sum_{\mathbf{z} \in \mathcal{Z}} \frac{q(\mathbf{z})}{q(\mathbf{z})} p_\theta(\mathbf{x}, \mathbf{z}) = \mathbb{E}_{\mathbf{z} \sim q(\mathbf{z})} \left[\frac{p_\theta(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \right]$$

Monte Carlo to the rescue:

1. Sample $\mathbf{z}^{(1)}, \cdots, \mathbf{z}^{(k)}$ from $q(\mathbf{z})$
2. Approximate expectation with sample average (*unbiased* estimator):

$$p_\theta(\mathbf{x}) \approx \frac{1}{k} \sum_{j=1}^{k} \frac{p_\theta(\mathbf{x}, \mathbf{z}^{(j)})}{q(\mathbf{z}^{(j)})}$$

Recall that for training, we need the *log*-likelihood $\log(p_\theta(\mathbf{x}))$. We could estimate it as:

$$\log(p_\theta(\mathbf{x})) \approx \log\left(\frac{1}{k} \sum_{j=1}^{k} \frac{p_\theta(\mathbf{x}, \mathbf{z}^{(j)})}{q(\mathbf{z}^{(j)})} \right) \stackrel{k=1}{\approx} \log\left(\frac{p_\theta(\mathbf{x}, \mathbf{z}^{(1)})}{q(\mathbf{z}^{(1)})} \right)$$

However, it's clear that $\mathbb{E}_{\mathbf{z}^{(1)} \sim q(\mathbf{z})} \left[\log\left(\frac{p_\theta(\mathbf{x}, \mathbf{z}^{(1)})}{q(\mathbf{z}^{(1)})} \right) \right] \neq \log\left(\mathbb{E}_{\mathbf{z}^{(1)} \sim q(\mathbf{z})} \left[\frac{p_\theta(\mathbf{x}, \mathbf{z}^{(1)})}{q(\mathbf{z}^{(1)})} \right] \right)$

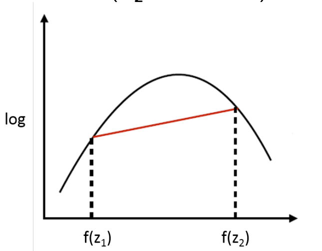
Evidence Lower Bound

Log-Likelihood function for Partially Observed Data is hard to compute:

$$\log\left(\sum_{\mathbf{z}\in\mathcal{Z}} p_\theta(\mathbf{x},\mathbf{z})\right) = \log\left(\sum_{\mathbf{z}\in\mathcal{Z}} \frac{q(\mathbf{z})}{q(\mathbf{z})} p_\theta(\mathbf{x},\mathbf{z})\right) = \log\left(\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right]\right)$$

- log() is a concave function. $\log(px + (1-p)x') \geq p\log(x) + (1-p)\log(x')$.
- Idea: use Jensen Inequality (for concave functions)

$$\log\left(\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}[f(\mathbf{z})]\right) = \log\left(\sum_{\mathbf{z}} q(\mathbf{z})f(\mathbf{z})\right) \geq \sum_{\mathbf{z}} q(\mathbf{z})\log f(\mathbf{z})$$



Evidence Lower Bound

Log-Likelihood function for Partially Observed Data is hard to compute:

$$\log\left(\sum_{\mathbf{z}\in\mathcal{Z}} p_\theta(\mathbf{x},\mathbf{z})\right) = \log\left(\sum_{\mathbf{z}\in\mathcal{Z}} \frac{q(\mathbf{z})}{q(\mathbf{z})} p_\theta(\mathbf{x},\mathbf{z})\right) = \log\left(\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right]\right)$$

- $\log()$ is a concave function. $\log(px+(1-p)x') \geq p\log(x)+(1-p)\log(x')$.
- Idea: use Jensen Inequality (for concave functions)

$$\log(\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[f(\mathbf{z})\right]) = \log(\sum_{\mathbf{z}} q(\mathbf{z})f(\mathbf{z})) \geq \sum_{\mathbf{z}} q(\mathbf{z})\log f(\mathbf{z}) = \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}[\log f(\mathbf{z})]$$

Choosing $f(\mathbf{z}) = \frac{p_\theta(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}$

$$\log\left(\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right]\right) \geq \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log\left(\frac{p_\theta(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right)\right]$$

Called Evidence Lower Bound (**ELBO**).

Variational inference

- Suppose $q(\mathbf{z})$ is **any** probability distribution over the hidden variables
- **Evidence lower bound** (ELBO) holds for any q

$$\begin{aligned}
\log p(\mathbf{x};\theta) &\geq \sum_{\mathbf{z}} q(\mathbf{z}) \log \left(\frac{p_\theta(\mathbf{x},\mathbf{z})}{q(\mathbf{z})} \right) \\
&= \sum_{\mathbf{z}} q(\mathbf{z}) \log p_\theta(\mathbf{x},\mathbf{z}) \underbrace{- \sum_{\mathbf{z}} q(\mathbf{z}) \log q(\mathbf{z})}_{\text{Entropy } H(q) \text{ of } q} \\
&= \sum_{\mathbf{z}} q(\mathbf{z}) \log p_\theta(\mathbf{x},\mathbf{z}) + H(q)
\end{aligned}$$

- Equality holds if $q = p(\mathbf{z}|\mathbf{x};\theta)$

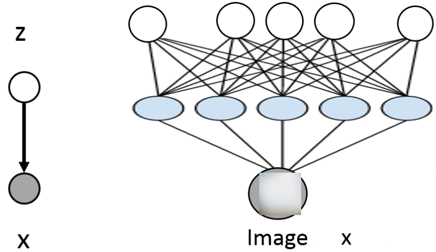
$$\log p(\mathbf{x};\theta) {\color{red}=} \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z},\mathbf{x};\theta) + H(q)$$

- (Aside: This is what we compute in the E-step of the EM algorithm)

Plan for today

1. Latent Variable Models
   - Learning deep generative models
   - Stochastic optimization:
     - Reparameterization trick
   - Inference Amortization

Variational Autoencoder



A mixture of an infinite number of Gaussians:

1. $\mathbf{z} \sim \mathcal{N}(0, I)$
2. $p(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}\left(\mu_\theta(\mathbf{z}), \Sigma_\theta(\mathbf{z})\right)$ where μ_θ,Σ_θ are neural networks
3. Even though $p(\mathbf{x} \mid \mathbf{z})$ is simple, the marginal $p(\mathbf{x})$ is very complex/flexible

Recap

- Latent Variable Models
  - Allow us to define complex models $p(\mathbf{x})$ in terms of simple building blocks $p(\mathbf{x} \mid \mathbf{z})$
  - Natural for unsupervised learning tasks (clustering, unsupervised representation learning, etc.)
  - No free lunch: much more difficult to learn compared to fully observed, autoregressive models because $p(\mathbf{x})$ is hard to evaluate (and optimize)

Variational inference

- Suppose $q(\mathbf{z})$ is **any** probability distribution over the hidden variables
- **Evidence lower bound** (ELBO) holds for any q

$$
\begin{aligned}
\log p(\mathbf{x};\theta) &\geq \sum_{\mathbf{z}} q(\mathbf{z}) \log \left(\frac{p_\theta(\mathbf{x},\mathbf{z})}{q(\mathbf{z})} \right) \\
&= \underbrace{\sum_{\mathbf{z}} q(\mathbf{z}) \log p_\theta(\mathbf{x},\mathbf{z})}_{\text{Loglikelihood as if fully observed}} \underbrace{- \sum_{\mathbf{z}} q(\mathbf{z}) \log q(\mathbf{z})}_{\text{Entropy } H(q) \text{ of } q} \\
&= \sum_{\mathbf{z}} q(\mathbf{z}) \log p_\theta(\mathbf{x},\mathbf{z}) + H(q)
\end{aligned}
$$

- Equality holds if $q = p(\mathbf{z}|\mathbf{x};\theta)$

$$
\log p(\mathbf{x};\theta) = \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z},\mathbf{x};\theta) + H(q)
$$

- (Aside: This is what we compute in the E-step of the EM algorithm)

Variational Inference

- Suppose $q(\mathbf{z})$ is **any** probability distribution over the hidden variables. A little bit of algebra reveals

$$D_{KL}(q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x};\theta)) = -\sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z},\mathbf{x};\theta) + \log p(\mathbf{x};\theta) - H(q) \geq 0$$

- **Evidence lower bound** (ELBO) holds for any q

$$\log p(\mathbf{x};\theta) \geq \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z},\mathbf{x};\theta) + H(q)$$

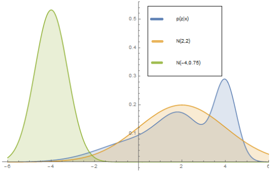
- Equality holds if $q = p(\mathbf{z}|\mathbf{x};\theta)$ because $D_{KL}(q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x};\theta)) = 0$

$$\log p(\mathbf{x};\theta) = \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z},\mathbf{x};\theta) + H(q)$$

- Confirms our intuition that we seek likely completions $\mathbf{z}$ given the observed values (evidence) $\mathbf{x}$.

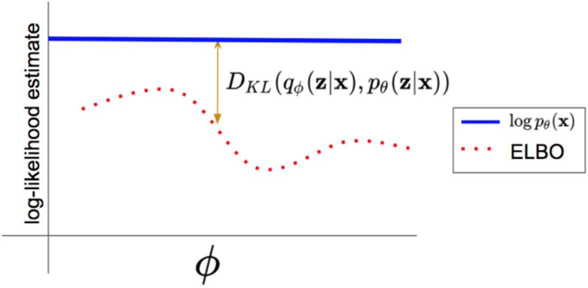
Intractable Posteriors

- What if the posterior $p(\mathbf{z}|\mathbf{x};\theta)$ is intractable to compute? In a VAE this corresponds to "inverting " the neural networks μ_θ,Σ_θ defining $p(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mu_\theta(\mathbf{z}), \Sigma_\theta(\mathbf{z}))$
- Suppose $q(\mathbf{z};\phi)$ is a (tractable) probability distribution over the hidden variables parameterized by ϕ (variational parameters)
  - For example, a Gaussian with mean and covariance specified by ϕ
$$q(\mathbf{z};\phi) = \mathcal{N}(\phi_1, \phi_2)$$
- **Variational inference**: pick ϕ so that $q(\mathbf{z};\phi)$ is as close as possible to $p(\mathbf{z}|\mathbf{x};\theta)$.



In the figure, the posterior $p(\mathbf{z}|\mathbf{x};\theta)$ (blue) is better approximated by $\mathcal{N}(2,2)$ (orange) than $\mathcal{N}(-4, 0.75)$ (green)

The Evidence Lower bound

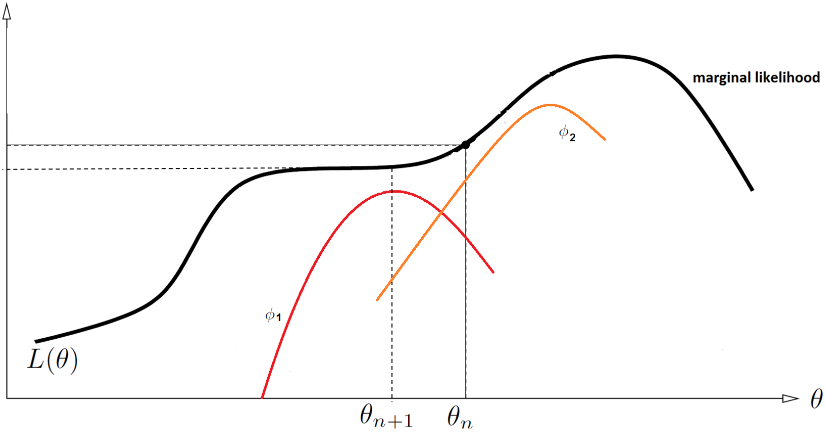


$$\log p(\mathbf{x};\theta) \geq \sum_{\mathbf{z}} q(\mathbf{z};\phi)\log p(\mathbf{z},\mathbf{x};\theta) + H(q(\mathbf{z};\phi)) = \underbrace{\mathcal{L}(\mathbf{x};\theta,\phi)}_{\text{ELBO}}$$

$$\log p(\mathbf{x};\theta) = \mathcal{L}(\mathbf{x};\theta,\phi) + D_{KL}(q(\mathbf{z};\phi)\|p(\mathbf{z}|\mathbf{x};\theta))$$

The better $q(\mathbf{z};\phi)$ can approximate the posterior $p(\mathbf{z}|\mathbf{x};\theta)$, the smaller $D_{KL}(q(\mathbf{z};\phi)\|p(\mathbf{z}|\mathbf{x};\theta))$ we can achieve, the closer ELBO will be to $\log p(\mathbf{x};\theta)$. Next: jointly optimize over θ and ϕ to maximize the ELBO over a dataset

Variational learning



$\mathcal{L}(\mathbf{x}; \theta, \phi_1)$ and $\mathcal{L}(\mathbf{x}; \theta, \phi_2)$ are both lower bounds. We want to jointly optimize θ and ϕ

The Evidence Lower bound applied to the entire dataset

- Evidence lower bound (ELBO) holds for any $q(\mathbf{z};\phi)$

$$\log p(\mathbf{x};\theta) \geq \sum_{\mathbf{z}} q(\mathbf{z};\phi) \log p(\mathbf{z},\mathbf{x};\theta) + H(q(\mathbf{z};\phi)) = \underbrace{\mathcal{L}(\mathbf{x};\theta,\phi)}_{\text{ELBO}}$$

- Maximum likelihood learning (over the entire dataset):

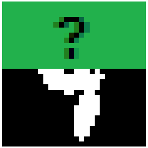
$$\ell(\theta;\mathcal{D}) = \sum_{\mathbf{x}^i \in \mathcal{D}} \log p(\mathbf{x}^i;\theta) \geq \sum_{\mathbf{x}^i \in \mathcal{D}} \mathcal{L}(\mathbf{x}^i;\theta,\phi^i)$$

- Therefore

$$\max_{\theta} \ell(\theta;\mathcal{D}) \geq \max_{\theta,\phi^1,\cdots,\phi^M} \sum_{\mathbf{x}^i \in \mathcal{D}} \mathcal{L}(\mathbf{x}^i;\theta,\phi^i)$$

- Note that we use different *variational parameters* ϕ^i for every data point $\mathbf{x}^i$, because the true posterior $p(\mathbf{z}|\mathbf{x}^i;\theta)$ is different across datapoints $\mathbf{x}^i$

A variational approximation to the posterior



- Assume $p(\mathbf{z}, \mathbf{x}; \theta)$ is close to $p_{\text{data}}(\mathbf{z}, \mathbf{x})$. $\mathbf{z}$ denotes the top half of the image (assumed to be latent)
- Suppose $q(\mathbf{z}; \phi)$ is a (tractable) probability distribution over the hidden variables $\mathbf{z}$ parameterized by ϕ (variational parameters)

$$q(\mathbf{z}; \phi) = \prod_{\text{unobserved variables } z_i} (\phi_i)^{z_i} (1 - \phi_i)^{(1 - z_i)}$$

- Is $\phi_i = 0.5 \ \forall i$ a good approximation to the posterior $p(\mathbf{z}|\mathbf{x}; \theta)$? No
- Is $\phi_i = 1 \ \forall i$ a good approximation to the posterior $p(\mathbf{z}|\mathbf{x}; \theta)$? No
- Is $\phi_i \approx 1$ for pixels i corresponding to the top part of digit **9** a good approximation? Yes
- Note: not true if $p(\mathbf{z}, \mathbf{x}; \theta)$ is far from $p_{\text{data}}(\mathbf{z}, \mathbf{x})$, i.e., at the beginning of learning

Learning via stochastic variational inference (SVI)

- Optimize $\sum_{\mathbf{x}^i \in \mathcal{D}} \mathcal{L}(\mathbf{x}^i; \theta, \phi^i)$ as a function of $\theta, \phi^1, \cdots, \phi^M$ using (stochastic) gradient descent

$$
\begin{aligned}
\mathcal{L}(\mathbf{x}^i; \theta, \phi^i) &= \sum_{\mathbf{z}} q(\mathbf{z}; \phi^i) \log p(\mathbf{z}, \mathbf{x}^i; \theta) + H(q(\mathbf{z}; \phi^i)) \\
&= E_{q(\mathbf{z}; \phi^i)}[\log p(\mathbf{z}, \mathbf{x}^i; \theta) - \log q(\mathbf{z}; \phi^i)]
\end{aligned}
$$

1. Initialize $\theta, \phi^1, \cdots, \phi^M$
2. Randomly sample a data point $\mathbf{x}^i$ from $\mathcal{D}$
3. Optimize $\mathcal{L}(\mathbf{x}^i; \theta, \phi^i)$ as a function of ϕ^i:
   1. Repeat $\phi^i = \phi^i + \eta \nabla_{\phi^i} \mathcal{L}(\mathbf{x}^i; \theta, \phi^i)$
   2. until convergence to $\phi^{i,*} \approx \arg\max_{\phi} \mathcal{L}(\mathbf{x}^i; \theta, \phi)$
4. Compute $\nabla_{\theta} \mathcal{L}(\mathbf{x}^i; \theta, \phi^{i,*})$
5. Update θ in the gradient direction. Go to step 2

- How to compute the gradients? There might not be a closed form solution for the expectations. So we use Monte Carlo sampling

Learning Deep Generative models

$$\begin{aligned}\mathcal{L}(\mathbf{x};\theta,\phi) &= \sum_{\mathbf{z}} q(\mathbf{z};\phi)\log p(\mathbf{z},\mathbf{x};\theta) + H(q(\mathbf{z};\phi)) \\ &= E_{q(\mathbf{z};\phi)}[\log p(\mathbf{z},\mathbf{x};\theta) - \log q(\mathbf{z};\phi)]\end{aligned}$$

- Note: dropped i superscript from ϕ^i for compactness
- To *evaluate* the bound, sample $\mathbf{z}^1, \cdots, \mathbf{z}^K$ from $q(\mathbf{z};\phi)$ and estimate

$$E_{q(\mathbf{z};\phi)}[\log p(\mathbf{z},\mathbf{x};\theta) - \log q(\mathbf{z};\phi)] \approx \frac{1}{K}\sum_k \log p(\mathbf{z}^k,\mathbf{x};\theta) - \log q(\mathbf{z}^k;\phi))$$

- Key assumption: $q(\mathbf{z};\phi)$ is tractable, i.e., easy to sample from and evaluate
- Want to compute $\nabla_\theta \mathcal{L}(\mathbf{x};\theta,\phi)$ and $\nabla_\phi \mathcal{L}(\mathbf{x};\theta,\phi)$
- The gradient with respect to θ is easy

$$\begin{aligned}\nabla_\theta E_{q(\mathbf{z};\phi)}[\log p(\mathbf{z},\mathbf{x};\theta) - \log q(\mathbf{z};\phi)] &= E_{q(\mathbf{z};\phi)}[\nabla_\theta \log p(\mathbf{z},\mathbf{x};\theta)] \\ &\approx \frac{1}{K}\sum_k \nabla_\theta \log p(\mathbf{z}^k,\mathbf{x};\theta)\end{aligned}$$

Learning Deep Generative models

$$
\begin{aligned}
\mathcal{L}(\mathbf{x};\theta,\phi) &= \sum_{\mathbf{z}} q(\mathbf{z};\phi)\log p(\mathbf{z},\mathbf{x};\theta) + H(q(\mathbf{z};\phi)) \\
&= E_{q(\mathbf{z};\phi)}[\log p(\mathbf{z},\mathbf{x};\theta) - \log q(\mathbf{z};\phi)]
\end{aligned}
$$

- Want to compute $\nabla_\theta \mathcal{L}(\mathbf{x};\theta,\phi)$ and $\nabla_\phi \mathcal{L}(\mathbf{x};\theta,\phi)$
- The gradient with respect to ϕ is more complicated because the expectation depends on ϕ
- We still want to estimate with a Monte Carlo average
- Later in the course we'll see a general technique called REINFORCE (from reinforcement learning)
- For now, a better but less general alternative that only works for continuous $\mathbf{z}$ (and only some distributions)

Reparameterization

- Want to compute a gradient with respect to ϕ of

$$E_{q(\mathbf{z};\phi)}[r(\mathbf{z})] = \int q(\mathbf{z};\phi) r(\mathbf{z}) d\mathbf{z}$$

  where $\mathbf{z}$ is now **continuous**

- Suppose $q(\mathbf{z};\phi) = \mathcal{N}(\mu, \sigma^2 I)$ is Gaussian with parameters $\phi = (\mu, \sigma)$. These are equivalent ways of sampling:
  - Sample $\mathbf{z} \sim q(\mathbf{z};\phi)$
  - Sample $\epsilon \sim \mathcal{N}(0, I)$, $\mathbf{z} = \mu + \sigma\epsilon = g(\epsilon;\phi)$. g is deterministic!
- Using this equivalence we compute the expectation in two ways:

$$E_{\mathbf{z}\sim q(\mathbf{z};\phi)}[r(\mathbf{z})] = \int q(\mathbf{z};\phi) r(\mathbf{z}) d\mathbf{z} = E_{\epsilon\sim\mathcal{N}(0,I)}[r(g(\epsilon;\phi))] = \int \mathcal{N}(\epsilon) r(\mu + \sigma\epsilon) d\epsilon$$

$$\nabla_\phi E_{q(\mathbf{z};\phi)}[r(\mathbf{z})] = \nabla_\phi E_\epsilon[r(g(\epsilon;\phi))] = E_\epsilon[\nabla_\phi r(g(\epsilon;\phi))]$$

- Easy to estimate via Monte Carlo if r and g are differentiable w.r.t. ϕ and ϵ is easy to sample from (backpropagation)
- $E_\epsilon[\nabla_\phi r(g(\epsilon;\phi))] \approx \frac{1}{K}\sum_k \nabla_\phi r(g(\epsilon^k;\phi))$ where $\epsilon^1, \cdots, \epsilon^K \sim \mathcal{N}(0, I)$.
- Typically much lower variance than REINFORCE

Learning Deep Generative models

$$
\begin{aligned}
\mathcal{L}(\mathbf{x};\theta,\phi) &= \sum_{\mathbf{z}} q(\mathbf{z};\phi)\log p(\mathbf{z},\mathbf{x};\theta) + H(q(\mathbf{z};\phi)) \\
&= E_{q(\mathbf{z};\phi)}[\underbrace{\log p(\mathbf{z},\mathbf{x};\theta) - \log q(\mathbf{z};\phi)}_{r(\mathbf{z},\phi)}]
\end{aligned}
$$

- Our case is slightly more complicated because we have $E_{q(\mathbf{z};\phi)}[r(\mathbf{z},\phi)]$ instead of $E_{q(\mathbf{z};\phi)}[r(\mathbf{z})]$. Term inside the expectation also depends on ϕ.
- Can still use reparameterization. Assume $\mathbf{z} = \mu + \sigma\epsilon = g(\epsilon;\phi)$ like before. Then

$$
\begin{aligned}
E_{q(\mathbf{z};\phi)}[r(\mathbf{z},\phi)] &= E_{\epsilon}[r(g(\epsilon;\phi),\phi)] \\
&\approx \frac{1}{K}\sum_{k} r(g(\epsilon^k;\phi),\phi)
\end{aligned}
$$

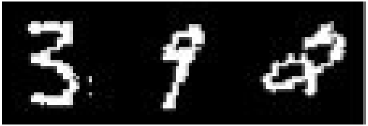
and use chain rule for the gradient.

Amortized Inference

$$\max_{\theta} \ell(\theta; \mathcal{D}) \geq \max_{\theta, \phi^1, \cdots, \phi^M} \sum_{\mathbf{x}^i \in \mathcal{D}} \mathcal{L}(\mathbf{x}^i; \theta, \phi^i)$$

- So far we have used a set of variational parameters ϕ^i for each data point $\mathbf{x}^i$. Does not scale to large datasets.
- **Amortization:** Now we learn a **single** parametric function f_λ that maps each $\mathbf{x}$ to a set of (good) variational parameters. Like doing regression on $\mathbf{x}^i \mapsto \phi^{i,*}$
  - For example, if $q(\mathbf{z}|\mathbf{x}^i)$ are Gaussians with different means $\mu^1, \cdots, \mu^m$, we learn a **single** neural network f_λ mapping $\mathbf{x}^i$ to μ^i
- We approximate the posteriors $q(\mathbf{z}|\mathbf{x}^i)$ using this distribution $q_\lambda(\mathbf{z}|\mathbf{x})$

A variational approximation to the posterior



- Assume $p(\mathbf{z}, \mathbf{x}^i; \theta)$ is close to $p_{\mathrm{data}}(\mathbf{z}, \mathbf{x}^i)$. Suppose $\mathbf{z}$ captures information such as the digit identity (label), style, etc.
- Suppose $q(\mathbf{z}; \phi^i)$ is a (tractable) probability distribution over the hidden variables $\mathbf{z}$ parameterized by ϕ^i
- For each $\mathbf{x}^i$, need to find a good $\phi^{i,*}$ (via optimization, expensive).
- **Amortized inference**: *learn* how to map $\mathbf{x}^i$ to a good set of parameters ϕ^i via $q(\mathbf{z}; f_\lambda(\mathbf{x}^i))$. f_λ learns how to solve the optimization problem for you
- In the literature, $q(\mathbf{z}; f_\lambda(\mathbf{x}^i))$ often denoted $q_\phi(\mathbf{z}|\mathbf{x})$

Learning with amortized inference

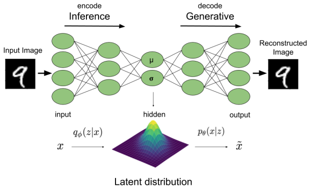
- Optimize $\sum_{\mathbf{x}^i \in \mathcal{D}} \mathcal{L}(\mathbf{x}^i; \theta, \phi)$ as a function of θ, ϕ using (stochastic) gradient descent

$$
\begin{aligned}
\mathcal{L}(\mathbf{x}; \theta, \phi) &= \sum_{\mathbf{z}} q_\phi(\mathbf{z}|\mathbf{x}) \log p(\mathbf{z}, \mathbf{x}; \theta) + H(q_\phi(\mathbf{z}|\mathbf{x})) \\
&= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log q_\phi(\mathbf{z}|\mathbf{x}))]
\end{aligned}
$$

1. Initialize $\theta^{(0)}, \phi^{(0)}$
2. Randomly sample a data point $\mathbf{x}^i$ from $\mathcal{D}$
3. Compute $\nabla_\theta \mathcal{L}(\mathbf{x}^i; \theta, \phi)$ and $\nabla_\phi \mathcal{L}(\mathbf{x}^i; \theta, \phi)$
4. Update θ, ϕ in the gradient direction

- How to compute the gradients? Use reparameterization like before

Autoencoder perspective



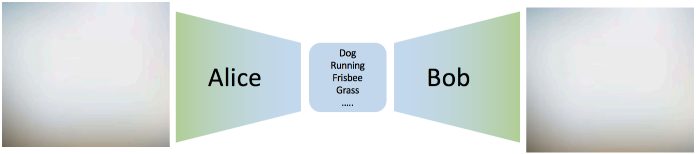
$$\begin{aligned}
\mathcal{L}(\mathbf{x};\theta,\phi) &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z},\mathbf{x};\theta) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\
&= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z},\mathbf{x};\theta) - \log p(\mathbf{z}) + \log p(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\
&= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z};\theta)] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))
\end{aligned}$$

1. Take a data point $\mathbf{x}^i$, map it to $\hat{\mathbf{z}}$ by sampling from $q_\phi(\mathbf{z}|\mathbf{x}^i)$ (*encoder*). Sample from a Gaussian with parameters $(\mu,\sigma) = encoder_\phi(\mathbf{x}^i)$
2. Reconstruct $\hat{\mathbf{x}}$ by sampling from $p(\mathbf{x}|\hat{\mathbf{z}};\theta)$ (*decoder*). Sample from a Gaussian with parameters $decoder_\theta(\hat{\mathbf{z}})$

What does the training objective $\mathcal{L}(\mathbf{x};\theta,\phi)$ do?

- First term encourages $\hat{\mathbf{x}} \approx \mathbf{x}^i$ ($\mathbf{x}^i$ likely under $p(\mathbf{x}|\hat{\mathbf{z}};\theta)$). *Autoencoding loss*!
- Second term encourages $\hat{\mathbf{z}}$ to have a distribution similar to the prior $p(\mathbf{z})$

Autoencoder perspective



1. Alice goes on a space mission and needs to send images to Bob. Given an image $\mathbf{x}^i$, she (stochastically) compresses it using $\hat{\mathbf{z}} \sim q_\phi(\mathbf{z}|\mathbf{x}^i)$ obtaining a message $\hat{\mathbf{z}}$. Alice sends the message $\hat{\mathbf{z}}$ to Bob
2. Given $\hat{\mathbf{z}}$, Bob tries to reconstruct the image using $p(\mathbf{x}|\hat{\mathbf{z}};\theta)$

- This scheme works well if $E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z};\theta)]$ is large
- The term $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))$ forces the distribution over messages to have a specific shape $p(\mathbf{z})$. If Bob knows $p(\mathbf{z})$, he can generate realistic messages $\hat{\mathbf{z}} \sim p(\mathbf{z})$ and the corresponding image, as if he had received them from Alice!