CSC 412/2506:
Probabilistic Learning and Reasoning

Week 7: Variational Inference II & VAEs

Ohad Shamir

University of Toronto

Overview for the first hour

- Variational Inference
- ELBO and its properties
- Estimating gradients of the ELBO
  - Simple Monte Carlo
  - Reparameterization trick

Recap: Posterior Inference for Latent Variable Models

We encountered a few latent variable models (e.g. the trueskill model).

These models have a factorization $p(x, z) = p(z)p(x|z)$ where:

- x are the observations or data,
- z are the unobserved (latent) variables
- $p(z)$ is usually called the **prior**
- $p(x|z)$ is usually called the **likelihood**
- The conditional distribution of the unobserved variables given the observed variables (aka the **posterior**) is

$$p(z|x) = \frac{p(x, z)}{p(x)} = \frac{p(x, z)}{\int p(x, z)dz}$$

- We assume $p(x) = \int p(x, z)dz$ is hard to compute

Variational methods

Variational inference works as follows:

- Choose a tractable parametric distribution $q_\phi(z)$ with parameters ϕ. This distribution will be used to approximate $p(z|x)$.
  - For example, $q_\phi(z) = \mathcal{N}(z|\mu, \Sigma)$ where $\phi = (\mu, \Sigma)$.
- Formulate and try to minimize some notion of "distance" between $p(z|x)$ and $q_\phi(z)$ that can be efficiently estimated. Usually we will use the KL divergence $\mathrm{KL}(q_\phi(z)\|p(z|x))$.

ELBO: Evidence Lower Bound

- Evaluating $\mathrm{KL}(q_\phi(z)\|p(z|x))$ is generally intractable because of the $p(z|x)$ term.
- Instead, we will solve a different optimization problem: maximize the **evidence lower bound (ELBO)**.
- We will show that maximizing the ELBO is equivalent to minimizing $\mathrm{KL}(q_\phi(z)\|p(z|x))$.

ELBO: Evidence Lower Bound

$$
\begin{aligned}
\mathrm{KL}(q_\phi(z)\|p(z|x)) &= \mathop{\mathbb{E}}_{z\sim q_\phi} \log \frac{q_\phi(z)}{p(z|x)} \\
&= \mathop{\mathbb{E}}_{z\sim q_\phi} \left[\log \left(q_\phi(z) \cdot \frac{p(x)}{p(z,x)} \right) \right] \\
&= \mathop{\mathbb{E}}_{z\sim q_\phi} \left[\log \frac{q_\phi(z)}{p(z,x)} \right] + \mathop{\mathbb{E}}_{z\sim q_\phi} \log p(x) \\
&:= -\mathcal{L}(\phi) + \log p(x)
\end{aligned}
$$

Where $\mathcal{L}(\phi)$ is the **ELBO**:

$$
\mathcal{L}(\phi) = \mathop{\mathbb{E}}_{z\sim q_\phi} \Big[\log p(z,x) - \log q_\phi(z) \Big]
$$

Thus, maximizing ELBO is same as minimizing $\mathrm{KL}(q_\phi(z)\|p(z|x))$!

Why "Evidence Lower Bound"?

$$\mathrm{KL}(q_\phi(z)\|p(z|x)) = -\mathcal{L}(\phi) + \log p(x).$$

- Rearranging, we get

$$\mathcal{L}(\phi) + \mathrm{KL}(q_\phi(z)\|p(z|x)) = \log p(x)$$

- Because $\mathrm{KL}(q_\phi(z)\|p(z|x)) \geq 0$,

$$\mathcal{L}(\phi) \leq \underbrace{\log p(x)}_{\text{"evidence"}}$$

Two Formulations of ELBO

We showed:

$$\mathcal{L}(\phi) = \underbrace{\mathop{\mathbb{E}}_{z\sim q_\phi}\Big[\log p(z,x)\Big]}_{\text{expected log-joint}} + \underbrace{\mathop{\mathbb{E}}_{z\sim q_\phi}\Big[-\log q_\phi(z)\Big]}_{\text{entropy}}$$

$\rightarrow$ Maximizing ELBO is maximizing log-joint as well as entropy.

ELBO can also be written as:

$$\begin{aligned}\mathcal{L}(\phi) &= \mathbb{E}_{q_\phi(z)}\left[\log p(x,z) - \log q_\phi(z)\right] \\ &= \mathbb{E}_{q_\phi(z)}\left[\log p(x \mid z) + \log p(z) - \log q_\phi(z)\right] \\ &= \underbrace{\mathbb{E}_{q_\phi(z)}[\log p(x \mid z)]}_{\text{expected log-likelihood}} - \underbrace{\mathrm{KL}(q_\phi(z)\|p(z))}_{\text{KL to prior}}.\end{aligned}$$

$\rightarrow$ Maximizing ELBO is maximizing log-likelihood while minimizing KL to prior

We will use second formulation later on.

Maximizing ELBO

Recall: $\nabla\mathcal{L}(\phi)$ gives the direction of the steepest ascent of $\mathcal{L}(\phi)$.

Gradient descent (GD) methods: $\phi_{t+1} = \phi_t + s_t\nabla\mathcal{L}(\phi_t)$.

- We have that $\mathcal{L}(\phi) = \mathop{\mathbb{E}}_{z\sim q_\phi}\Big[\log p(x,z) - \log q_\phi(z)\Big]$.
- We need $\nabla_\phi\mathcal{L}(\phi)$ (or its unbiased estimate for stochastic GD).

Approximating the gradient of some $\mathbb{E}_Z(f(Z,\phi))$:

- If the distribution of Z independent of ϕ then

$$\nabla_\phi\mathbb{E}_Z(f(Z,\phi)) = \mathbb{E}_Z(\nabla_\phi f(Z,\phi)).$$

- We then have $\nabla_\phi\mathbb{E}_Z(f(Z,\phi)) \approx \frac{1}{n}\sum_{i=1}^n \nabla_\phi f(z_i,\phi)$ (as in simple Monte-Carlo)
- Problem: In our case the distribution of z depends on ϕ.

The reparameterization trick

Problem:

$$\nabla_\phi \mathop{\mathbb{E}}_{z\sim q_\phi}\Big[\log p(x,z) - \log q_\phi(z)\Big] \neq \mathop{\mathbb{E}}_{z\sim q_\phi}\Big[\nabla_\phi \left(\log p(x,z) - \log q_\phi(z)\right)\Big].$$

But in some situations there is a trick:

- We break this sampling process into two parts:
  - Sample a random variable ϵ that has fixed (or no) parameters, such as a uniform distribution or standard normal.
  - Deterministically compute z's as a function ϕ and ϵ, such that:
    - $\epsilon \sim p_0(\epsilon)$
    - $z = T(\epsilon, \phi)$
    - $\implies$
    - $z \sim q_\phi(z)$

The reparameterization trick

- For example, $z = \mu + \sigma\epsilon = T(\phi, \epsilon)$ (here $\phi = (\mu, \sigma)$).
  - $\epsilon \sim \mathcal{N}(0, 1)$
  - $z = \mu + \epsilon\sigma$
  - $\implies$
  - $z \sim \mathcal{N}(\mu, \sigma)$
- This makes the density independent of the parameter ϕ, which will let us use simple Monte Carlo: $z = T(\phi, \epsilon)$

$$\begin{aligned}
\nabla_\phi \mathcal{L}(\phi) &= \nabla_\phi \mathbb{E}_{z \sim q_\phi(z)} \Big[\log p(x, z) - \log q_\phi(z) \Big] \\
&= \nabla_\phi \mathbb{E}_{\epsilon \sim p_0(\epsilon)} \Big[\log p(x, T(\phi, \epsilon)) - \log q_\phi(T(\phi, \epsilon)) \Big] \\
&= \mathbb{E}_{\epsilon \sim p_0(\epsilon)} \nabla_\phi \Big[\log p(x, T(\phi, \epsilon)) - \log q_\phi(T(\phi, \epsilon)) \Big]
\end{aligned}$$

SVI: Stochastic Variational Inference

- Instead of computing the full gradient (which is in general not possible), we compute a simple Monte Carlo estimate of it.
- Specifically, instead of

$$\mathbb{E}_{\epsilon\sim p_0(\epsilon)}\nabla_\phi\Big[\log p(x,T(\phi,\epsilon))-\log q_\phi(T(\phi,\epsilon))\Big]$$

we work with a mini-batch of size m

$$\widehat{\mathbb{E}}_{\epsilon\sim p_0(\epsilon)}\nabla_\phi\Big[\log p(x,T(\phi,\epsilon))-\log q_\phi(T(\phi,\epsilon))\Big]$$
$$\approx\frac{1}{m}\sum_{i=1}^{m}\nabla_\phi\Big[\log p(x,T(\phi,\epsilon_i))-\log q_\phi(T(\phi,\epsilon_i))\Big]$$

MCMC vs. SVI

Pros of MCMC:

- Accurate results (at least asymptotically)
- Flexibility
- No approximation
- Handles multimodal distributions

Cons of MCMC:

- High computational cost
- Requires tuning of hyperparameters
- Convergence issues
- Inefficient in sampling complex dependencies

MCMC vs. SVI

Pros of SVI:

- Faster convergence
- Scalability
- Ease of use

Cons of SVI:

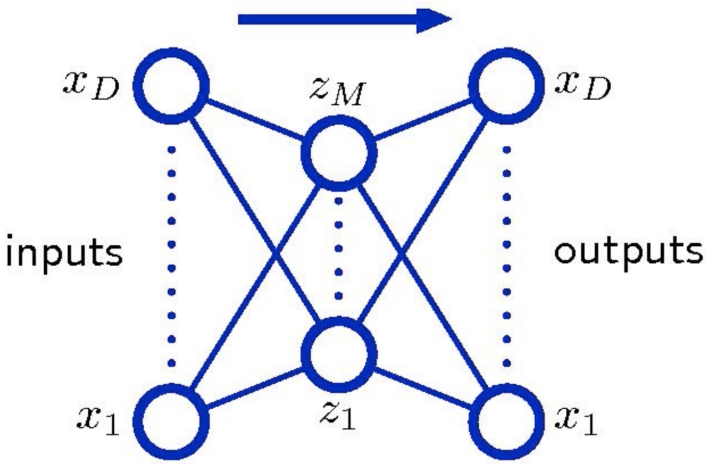
- Approximate results; can sometimes collapse to something degenerate
- Limited flexibility
- Sensitive to choice of hyperparameters

Will probably play more with it in assignment 3

Overview of rest of talk

- Autoencoders
- Variational Autoencoders
  (combines autoencoders with stuff we just learned)

Non-linear Dimension Reduction



- Neural networks can be used for nonlinear dimensionality reduction.
- This is achieved by having the same number of outputs as inputs. These models are called autoencoders.
- Consider a multilayer perceptron that has D inputs, D outputs, and M hidden units, with $M < D$.
- We can squeeze the information through some kind of bottleneck.
- If we use a linear network (linear activation) this is very similar to Principal Components Analysis.

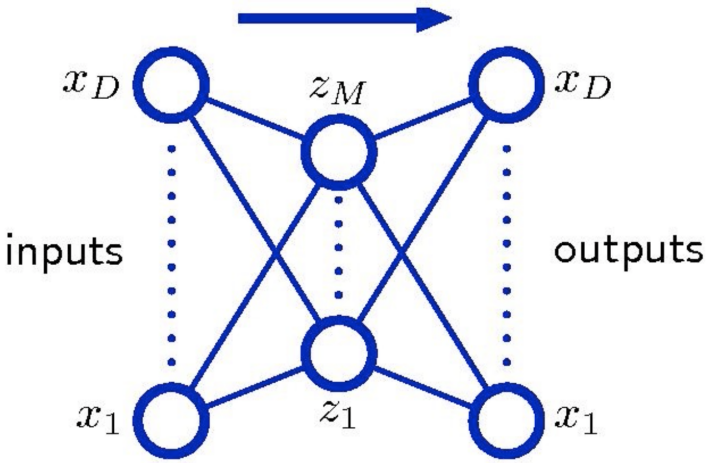
A 2-Layer Autoencoder

- Given an input $\boldsymbol{x}$, its corresponding reconstruction is given by:

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = \sum_{j=1}^{M} w_{kj}^{(2)} \sigma\Big(\sum_{i=1}^{D} w_{ji}^{(1)} x_i \Big), \quad k = 1, ..., D.$$

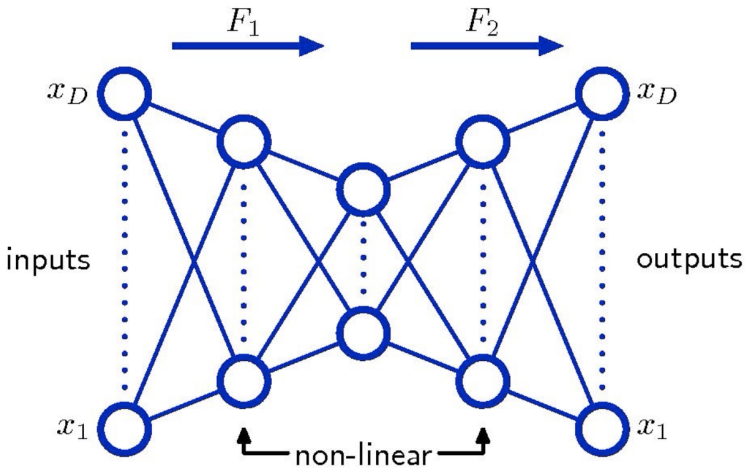
- We can determine the network parameters $\boldsymbol{w}$ by minimizing the reconstruction error:

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} \| y(\boldsymbol{x}_n, \boldsymbol{w}) - \boldsymbol{x}_n \|^2$$



- If the hidden and output layers are linear, essentially PCA (M hidden units will span the same space as the first M principal components).

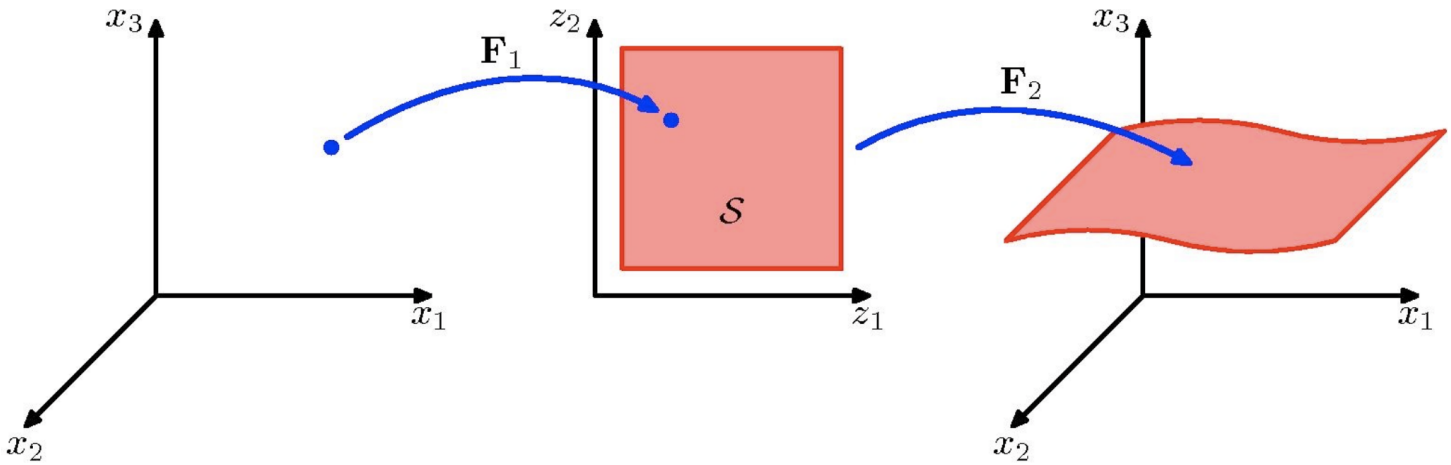
Deep Autoencoders



- We can put extra nonlinear hidden layers between the input and the bottleneck and between the bottleneck and the output.
- This gives nonlinear generalization of PCA, providing non-linear dimensionality reduction.
- The network can be trained by the minimization of the reconstruction error function.
- Trade-off: Deeper autoencoders can be harder to train.

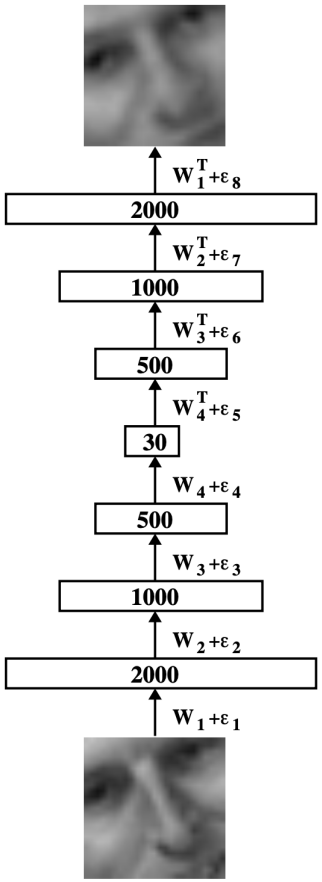
Geometrical Interpretation

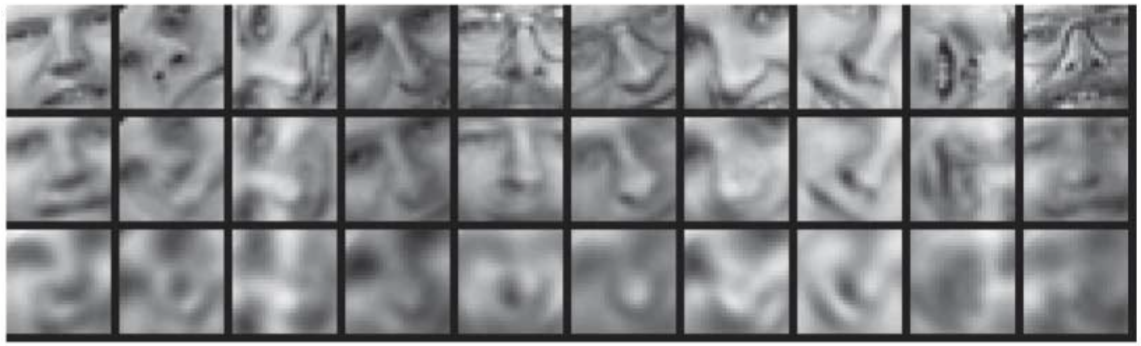
- Geometrical interpretation of the mappings performed by the network with 2 hidden layers for the case of $D = 3$ and $M = 2$ units in the middle layer.



- The mapping F_1 defines a nonlinear projection of points in the original D-space into the M-dimensional subspace.
- The mapping F_2 maps from an M-dimensional space into D-dimensional space.

Deep Autoencoders

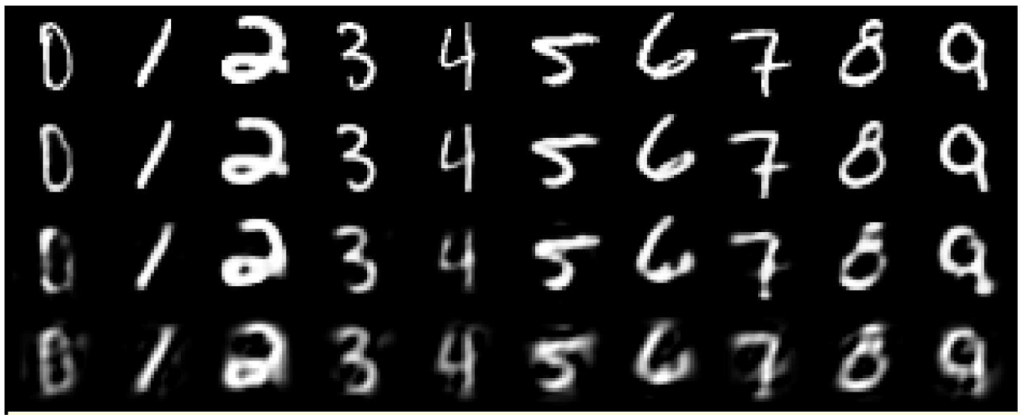




- We can consider very deep autoencoders.
- By row:
  - Real data
  - Deep autoencoder with a bottleneck of 30 linear units
  - 30-dimensional PCA

Deep Autoencoders

- Similar model for MNIST handwritten digits:



- Deep autoencoders produce much better reconstructions.

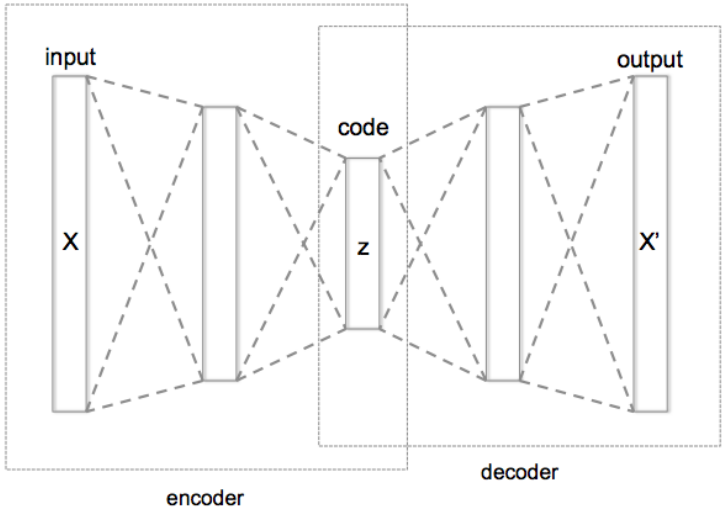
Autoencoders: Summary

Autoencoders reconstruct their input via an encoder and a decoder.

- **Encoder**: $g(x) = z \in F, \quad x \in X$
- **Decoder**: $f(z) = \tilde{x} \in X$
- where X is the data space, and F is the feature (latent) space.
- z is the code, compressed representation of the input, x. It is important that this code is a bottleneck, i.e. that

$$\dim F \ll \dim X$$

- Goal: $\tilde{x} = f(g(x)) \approx x$.



Issues with (deterministic) Autoencoders

- **Issue 1**: Proximity in data space does not mean proximity in feature space
  - The embeddings learned by the model are deterministic, i.e.

$$g(x_1) = z_1 \Rightarrow f(z_1) = \tilde{x}_1$$
$$g(x_2) = z_2 \Rightarrow f(z_2) = \tilde{x}_2$$

  - but proximity in feature space is not "directly" enforced for inputs in close proximity in data space, i.e.

$$x_1 \approx x_2 \not\Rightarrow z_1 \approx z_2$$

  - The latent space may not be continuous, or allow easy interpolation.

Issues with (deterministic) Autoencoders

- **Issue 1**: Proximity in data space does not mean proximity in feature space
  - If the space has discontinuities (eg. gaps between clusters) and you sample/generate from there, the decoder will simply generate an unrealistic output.

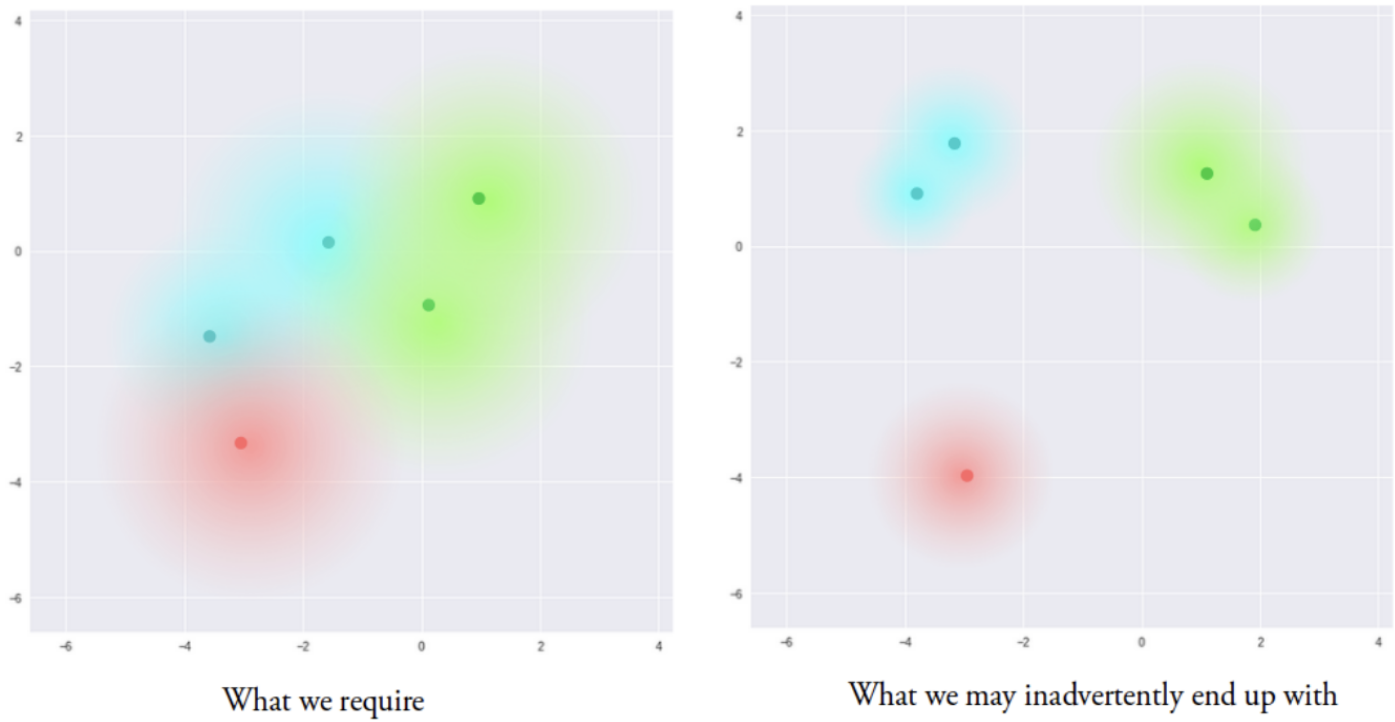
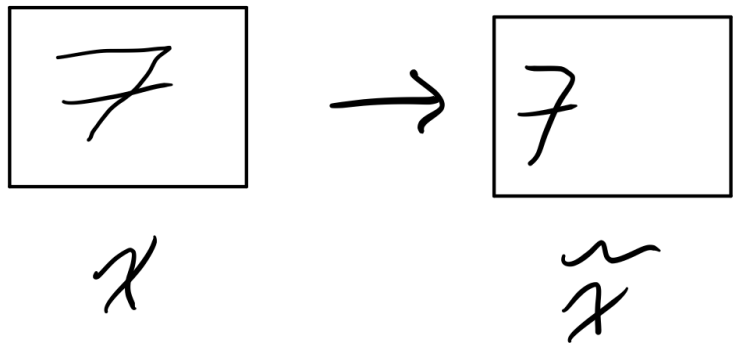


Image credit: I. Shafkat

Issues with (deterministic) Autoencoders

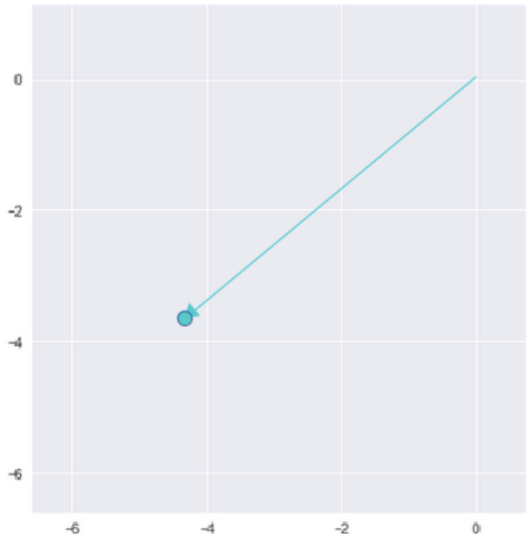
- **Issue 2**: How to measure the goodness of a reconstruction?



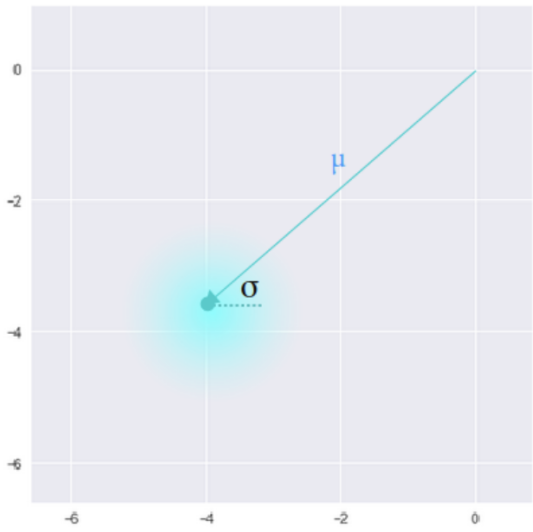
  - The reconstruction looks quite good. However, a simple distance metric such as difference between pixels will consider the images far apart.
  - Choosing an appropriate metric for evaluating model performance can be difficult

Variational Autoencoders

- Variational autoencoders (VAEs) encode inputs with uncertainty.
- Unlike standard autoencoders, the encoder of a VAE outputs a probability distribution, $q_\phi(z)$ to approximate $p(z|x)$.
- Instead of the encoder learning an encoding vector, it learns two vectors: vector of means, μ, and another vector of standard deviations, σ (one for each data point)



Standard Autoencoder
(direct encoding coordinates)



Variational Autoencoder
(μ and σ initialize a probability distribution)

VAE: Specifics

- VI idea: ELBO, where both p, q are parameterized distributions (by some θ, ϕ).
  ELBO can be written as

$$\mathcal{L}(\theta, \phi; x) = \mathbb{E}_{z \sim q_\phi}\Big[\log p_\theta(x|z)\Big] - KL(q_\phi(z)||p(z))$$

- The encoder and decoder in a VAE are:
  - **Encoder** (data point $\rightarrow$ distribution q_{ϕ_i}):
    $q_{\phi_i}(z) = q_{\phi_i}(z|x_i) = \mathcal{N}(\mu_i, \sigma_i^2)$ where $\phi_i = (\mu_i, \log \sigma_i)$
    - Note: σ_i encoded as $\log \sigma_i$ so that $\sigma_i > 0$ automatically enforced
  - **Decoder** (sample from $q_{\phi_i} \rightarrow$ a distribution parameter): $f(z_i) = \theta_i$ typically a neural network

VAE Pipeline

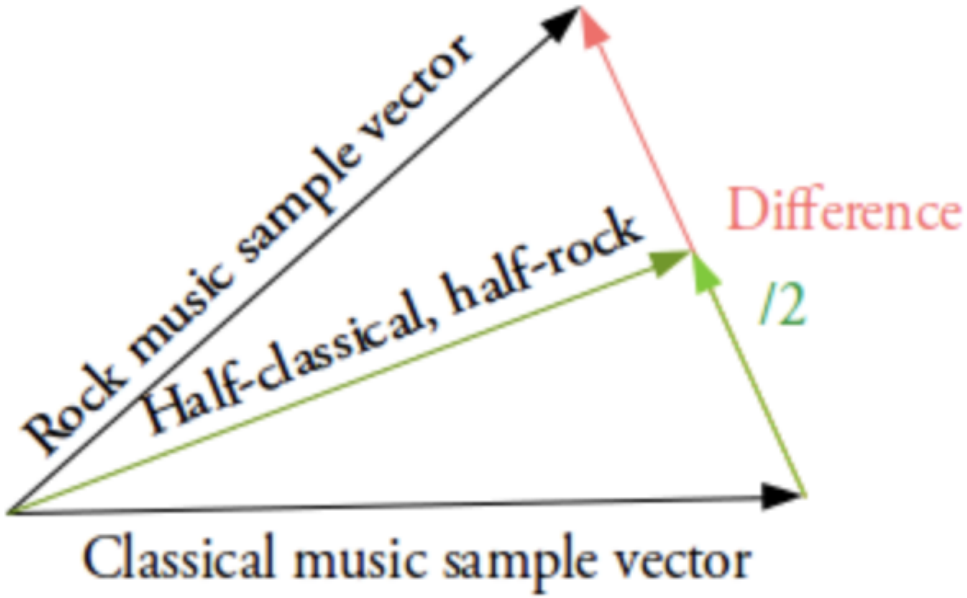
- For a given input (or minibatch) x_i,
  - Sample $z_i \sim q_{\phi_i}(z|x_i)$. This is the code in our feature space F.
  - Run the code through decoder and write the likelihood: $p_\theta(x|z)$.
  - Loss function is (negative of) ELBO:
    $$\mathcal{L}(x;\theta,\phi) = -E_{z_\phi \sim q_\phi}\Big[\log p_\theta(x|z)\Big] + KL(q_\phi(z|x)||p(z))$$
- Use gradient-based optimization to optimize over ϕ, θ.

After VAE is trained

- Once a VAE is trained, we can sample new inputs

$$z \sim p(z) \qquad \tilde{x} \sim p_\theta(x|z)$$

- We can also interpolate between inputs, using simple vector arithmetic.



Interpolating between samples

Example: MNIST

- We choose the prior on z to be the standard Gaussian

$$p(z) \sim \mathcal{N}(0, I)$$

- our likelihood function to be

$$p_\theta(x|z) = \text{Bernoulli}(\theta)$$

- and our approximate posterior is

$$q_{\phi_i}(z|x_i) = \mathcal{N}(\mu_i, \sigma_i^2 I)$$

- To get our reconstructed input, we simply evaluate

$$\tilde{x} \sim p_\theta(x|z)$$

- We will use neural networks as our encoder and decoder!

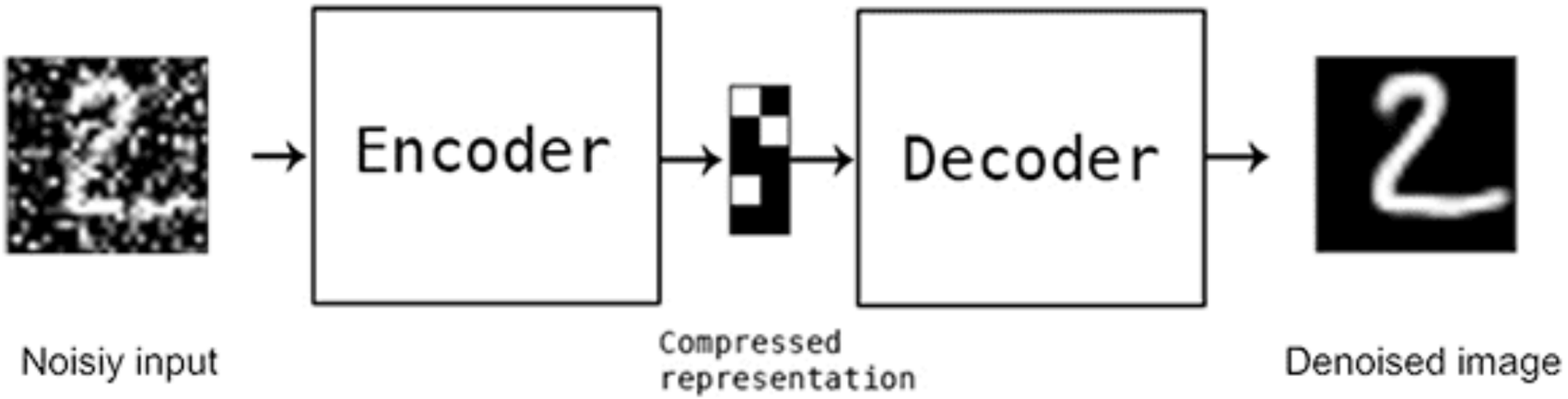
The Reparametrization Trick

- Encoder generates a code by sampling from $q_\phi(z|x)$.
- This sampling process introduces a problem: gradients are blocked from flowing into the encoder, and hence it will not train.
- To solve this problem, we use the **reparameterization trick**.
  - Instead of sampling z directly from its distribution (e.g. $z_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$) we express z_i as

$$z_i = \mu_i + \sigma_i \times \varepsilon_i \quad \text{where } \varepsilon_i \sim \mathcal{N}(0, I)$$

  with this, gradients can now flow through the entire network.

Application of Autoencoders: Image Denoising



- We can train a **denoising autoencoder**.
- Given noisy image, minimize the reconstruction error between the decoder output and clean image.
- Requires training on noisy-clean image pairs
- Alternative from earlier in course: Loopy BP. Works directly on a single noisy image, without training on a dataset.
  - However, still requires a probabilistic model of the image+noise process

Amortized Inference

- Instead of learning a separate ϕ_i for each data example, we'll just have a single global ϕ that specifies the parameters of a single "recognition model" $q_\phi(z_i|x_i)$.
- We can use a neural network to specify $q_\phi(z_i|x_i)$!
- For example

$$q_\phi(z_i|x_i) = \mathcal{N}(z_i|\mu_\phi(x_i), \Sigma_\phi(x_i))$$

- Allows to reduce the number of parameters (at the cost of making the encoder less expressive, as parameters are shared across data points)

VAE vs Amortized VAE Pipeline Summary

- For a given input (or minibatch) x_i,
  - **Standard VAE**
  - Sample $z_i \sim q_{\phi_i}(z|x_i) = \mathcal{N}(\mu_i, \sigma_i^2 I)$.
  - **Amortized VAE**
  - Sample $z_i \sim q_\phi(z|x_i) = \mathcal{N}(\mu_\phi(x_i), \Sigma_\phi(x_i))$
- Run the code through decoder and get likelihood: $p_\theta(x|z)$.
- Compute the loss function: $\mathcal{L}(x;\theta,\phi) = -E_{z_\phi \sim q_\phi}\Big[\log p_\theta(x|z)\Big] + KL(q_\phi(z|x)||p(z))$
- Use gradient-based optimization to backpropogate $\nabla_\theta L$, $\nabla_\phi L$ (using reparameterization trick where needed)

Example: MNIST

- We choose the prior on z to be the standard Gaussian

$$p(z) \sim \mathcal{N}(0, I)$$

- our likelihood function to be

$$p_\theta(x|z) = \text{Bernoulli}(\theta)$$

- and our approximate posterior is

$$q_\phi(z|x_i) = \mathcal{N}(\mu_\phi(x_i), \Sigma_\phi(x_i))$$

- Finally, we use neural networks as our encoder and decoder
  - **Encoder**: $g_\phi(x_i) = [\mu(x_i), \log \Sigma(x_i)]$
  - **Decoder**: $f_\theta(z_i) = \theta_i$
  - where θ_i are parameters of a Bernoulli rv for each input pixel.
- To get our reconstructed input, we simply evaluate

$$\tilde{x} \sim p_\theta(x|z)$$

MNIST: Autoencoder vs VAE

Codes generated by AE (left); VAE (right)

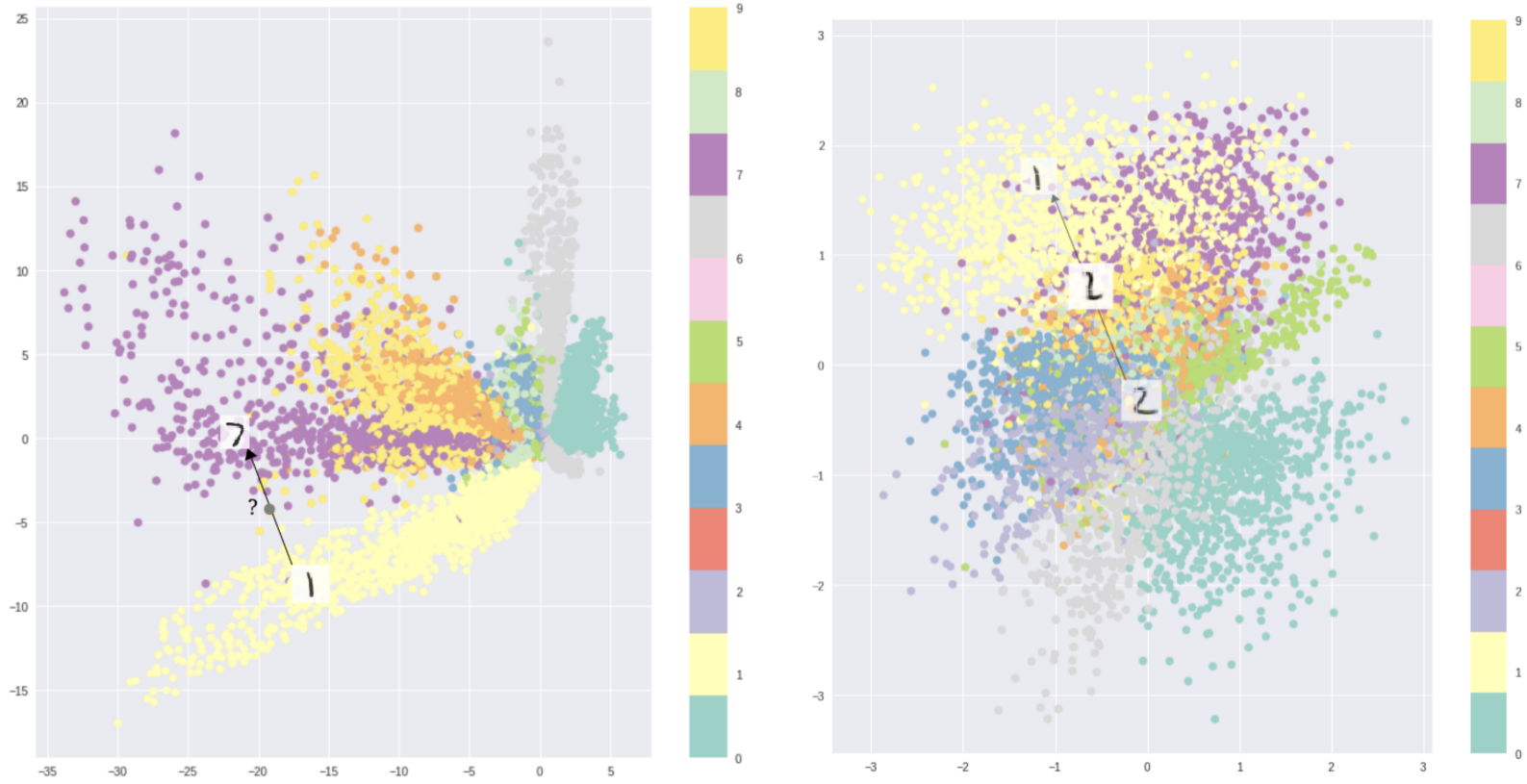


Image credit: I. Shafkat

Summary

- This lecture covered the basics of variational inference:
  - Elbo
  - Autoencoders
  - VAEs