

CSC 412/2506:
Probabilistic Learning and Reasoning
Week 9: Variational Inference II & VAEs

Murat A. Erdogdu

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 $\phi = (\mu, \Sigma)$.
- Encode some notion of "distance" between $p(z|x)$ and $q_\phi(z)$ that can be efficiently estimated. Usually we will use the KL divergence.
- Minimize this distance.

Recall: KL divergence

We will measure the difference between q and p using the **Kullback-Leibler divergence**

$$\begin{aligned}\text{KL}(q_\phi(z) \| p(z|x)) &= \int q_\phi(z) \log \frac{q_\phi(z)}{p(z|x)} dz \\ &= \mathbb{E}_{z \sim q_\phi} \log \frac{q_\phi(z)}{p(z|x)}\end{aligned}$$

ELBO: Evidence Lower Bound

- Evaluating $\text{KL}(q_\phi(z) \| p(z|x))$ is intractable because of the integral over z and the term $p(z|x)$, which is intractable to normalize.
- We can still “optimize” this KL without knowing the normalization constant $p(x)$.
- We solve a surrogate optimization problem: maximize the **evidence lower bound (ELBO)**.
- Maximizing the ELBO is equivalent to minimizing

$$\text{KL}(q_\phi(z) \| p(z|x)).$$

ELBO: Evidence Lower Bound

Maximizing the ELBO is the same as minimizing $\text{KL}(q_\phi(z)||p(z|x))$.

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

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

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

ELBO: Evidence Lower Bound

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

- Rearranging, we get

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

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

$$\mathcal{L}(\phi) \leq \log p(x)$$

- maximizing the ELBO \Rightarrow minimizing $\text{KL}(q_\phi(z) \| p(z|x))$.

- Note: $\mathcal{L}(\phi) = \mathbb{E}_{z \sim q_\phi} [\log p(z, x)] + \mathbb{E}_{z \sim q_\phi} [-\log q_\phi(z)]$, so

ELBO = **expected log-joint** + **entropy**

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

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

- If the distribution of Y independent of ϕ then

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

- We then have $\nabla_\phi \mathbb{E}(f(Y, \phi)) \approx \frac{1}{n} \sum_{i=1}^n \nabla_\phi f(y_i, \phi)$.
- Problem: In our case the distribution of z depends on ϕ .

The reparameterization trick

Problem:

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

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: $z = T(\phi, \epsilon)$

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

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.
- For example, instead of

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

we work with a mini-batch of size m

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

MCMC: Pros & Cons

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

SVI: Pros & Cons

Pros of SVI:

- Faster convergence
- Scalability
- Ease of use

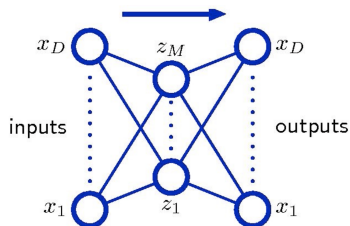
Cons of SVI:

- Approximate results
- Limited flexibility
- Mode seeking
- Sensitive to choice of hyperparameters

Overview of the second hour

- Autoencoders
- Variational Autoencoders

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.

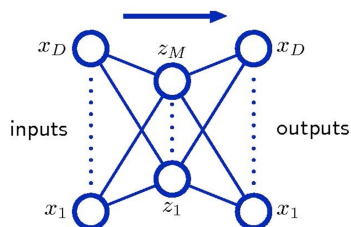
Autoencoders and PCA

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

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

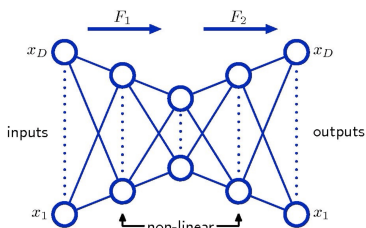
- We can determine the network parameters \mathbf{w} by minimizing the reconstruction error:

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



- If the hidden and output layers are linear, it will learn hidden units that are linear functions of the data and minimize squared error.
- M hidden units will span the same space as the first M principal components.

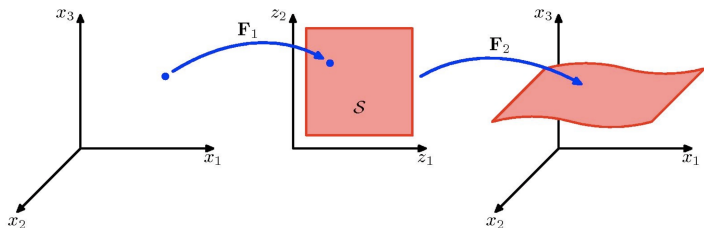
Deep Autoencoders



- 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.
- Much harder to train.

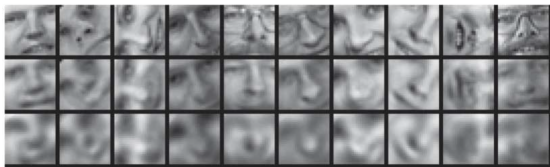
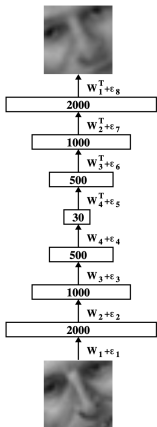
Geometrical Interpretation

- 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.

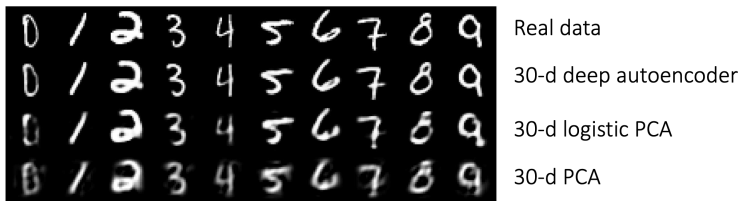
Deep Autoencoders



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

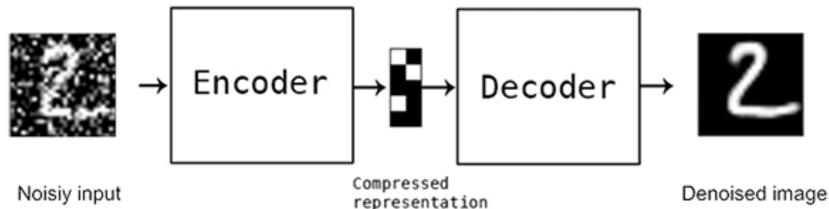
Deep Autoencoders

- Similar model for MNIST handwritten digits:



- Deep autoencoders produce much better reconstructions.

Application: Image Denoising



- We can train a **denoising autoencoder**.
- We feed noisy image as an input to the encoder
- Minimize the reconstruction error between the decoder output and original image.
- This method requires training and knowledge of the noise structure (fully supervised).
- In contrast, loopy BP works for a single noisy image and doesn't require the knowledge of noise structure (unsupervised).

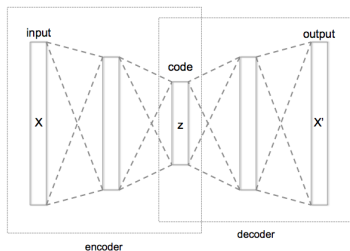
Autoencoders: Summary

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 codes learned by the model are deterministic, i.e.

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

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

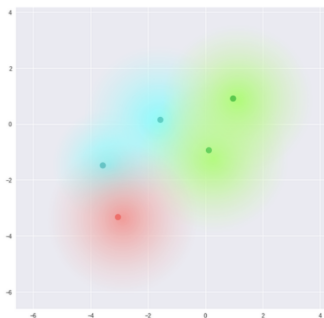
- ▶ but proximity in feature space is not “directly” enforced for inputs in close proximity in data space, i.e.

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

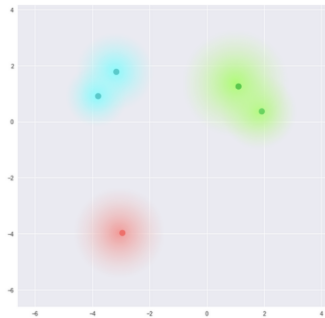
- ▶ The latent space may not be continuous, or allow easy interpolation.

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 a variation from there, the decoder will simply generate an unrealistic output.



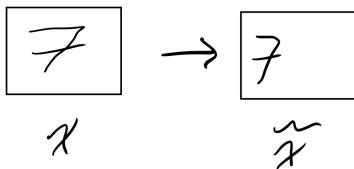
What we require



What we may inadvertently end up with

Issues with (deterministic) Autoencoders

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



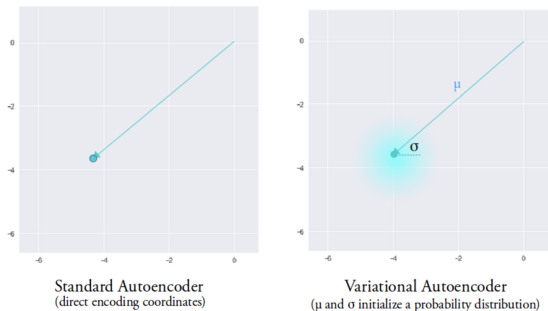
- ▶ The reconstruction looks quite good. However, if we chose a simple distance metric between inputs and reconstructions, we would heavily penalize the left-shift in the reconstruction \tilde{x} .
- ▶ Choosing an appropriate metric for evaluating model performance can be difficult, and that a miss-aligned objective can be disastrous.

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 .

Variational Autoencoders

- The mean μ controls where encoding of input is centered while the standard deviation controls how much can the encoding vary.



- Encodings are generated at random from the “circle”, the decoder learns that all nearby points refer to the same input.

Image credit: I. Shafkat

VAE: Specifics

- Our model is generated by the joint distribution over the latent codes and the input data $p(x, z)$. Decomposing

$$p(x, z) = \text{prior} \times \text{likelihood} = p(z)p(x|z)$$

- The encoder is $p(z|x) = p(x, z)/p(x)$.
- However, learning $p(x) = \int p(x|z)p(z)dz$ is intractable.
- We introduce an approximation with its own set of parameters, q_ϕ , and learn these parameters such that

$$q_\phi(z) \approx p(z|x).$$

VAE: Specifics

- VI idea:

$$\begin{aligned}\mathcal{L}(\theta, \phi; x) &= \text{ELBO} \\ &= \mathbb{E}_{z \sim q_\phi} \left[\log p_\theta(x|z) \right] - KL(q_\phi(z) || p(z))\end{aligned}$$

which is the (negative) loss function we use when training VAEs.

- First term is the expected log-likelihood and the second is the divergence of q_ϕ from the true prior.
- The encoder and decoder in a VAE become:
 - ▶ **Encoder:** $q_{\phi_i}(z) = q_{\phi_i}(z|x_i) = \mathcal{N}(\mu_i, \sigma_i^2)$ where $\phi_i = (\mu_i, \log \sigma_i)$
 - ▶ **Decoder:** $f(z_i) = \theta_i$ typically a neural network

VAE Pipeline

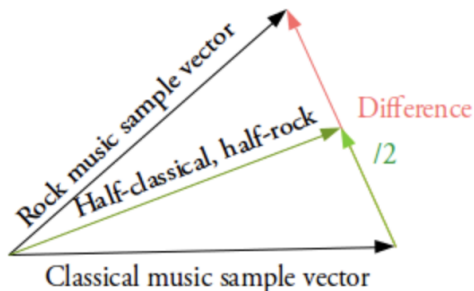
- For a given input (or minibatch) x_i ,
 - ▶ Sample $z_i \sim q_{\phi_i}(z|x_i)$. This is the code in our feature space F .
 - ▶ Run the code through decoder and write the likelihood: $p_{\theta}(x|z)$.
 - ▶ Compute the loss function:
$$\mathcal{L}(x; \theta, \phi) = -E_{z_{\phi} \sim q_{\phi}} \left[\log p_{\theta}(x|z) \right] + KL(q_{\phi}(z|x) || p(z))$$
- Use gradient-based optimization to backpropogate $\nabla_{\theta}L$, $\nabla_{\phi}L$

After VAE is trained

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

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

- 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 major 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.

Amortized Inference

- Instead of doing VI from scratch every time we see a new datapoint, we learn a function that can look at the data for a point x_i , and then output an approximate posterior $q_\phi(z_i|x_i)$. We'll call this a "recognition model"
- Instead of a separate ϕ_i for each data example, we'll just have a single global ϕ that specifies the parameters of the recognition model.
- Because the relationship between data and posteriors is complex and hard to specify by hand, we'll do this with a neural network!

Amortized Inference

- We can simply have a network take in x_i , and output the mean and variance vector for a Gaussian:
- Then the approximate posterior is given by

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

VAE vs Amortized VAE Pipeline

- 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:

$$L(x; \theta, \phi) = -E_{z_{\phi} \sim q_{\phi}} \left[\log p_{\theta}(x|z) \right] + KL(q_{\phi}(z|x) || p(z))$$

- Use gradient-based optimization to backpropogate $\nabla_{\theta} L, \nabla_{\phi} L$

Standard vs Amortized VAE

- This allows us to use the share parameters for all data points, and reduce the number of parameter for the encoder to that of the encoding NN.
- Standard VAE encoder is more expressive since no parameters are shared across different data points.

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

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

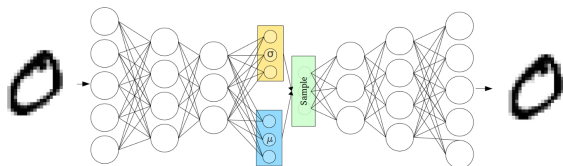
- and our approximate posterior is

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

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

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

Example: MNIST



- We use neural networks for both the encoder and the decoder.
- We compute the loss function $-\mathcal{L}(\theta, \phi; x)$ and propagate its derivative with respect to θ and ϕ , $\nabla_{\theta} L$, $\nabla_{\phi} L$, through the network during training.
- We need reparametrization trick as before!

MNIST: Autoencoder vs VAE

Codes generated by L: AE R: VAE

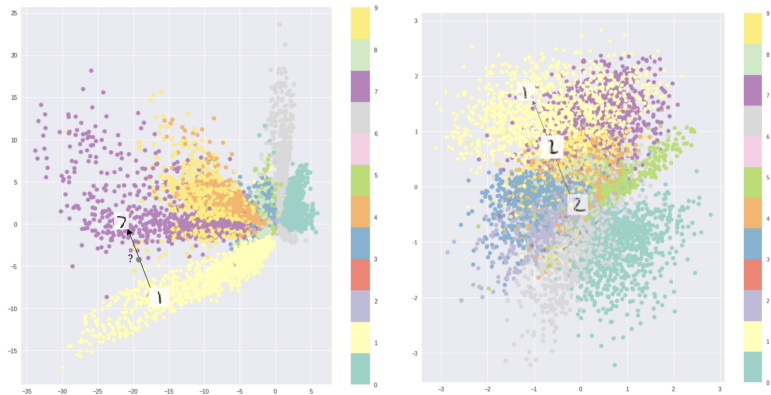


Image credit: I. Shafkat

VAE loss interpretation

- The VAE maximization objective can be written as

$$\begin{aligned}\mathcal{L}(x; \theta, \phi) &= E_{z_\phi \sim q_\phi} \left[\log p_\theta(x|z) \right] - KL(q_\phi(z|x) || p(z)) \\ &= E_{z_\phi \sim q_\phi} \left[\log p_\theta(x, z) \right] + H(q_\phi)\end{aligned}$$

- **Interpretation 1:** Maximize expected complete data log-likelihood while penalizing low entropy solutions.
- **Interpretation 2:** Maximize expected log-likelihood while penalizing solutions that are different from the prior.

Summary

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