Variational Autoencoders

Thibault Randrianarisoa

University of Toronto, Winter 2026

March 24, 2026

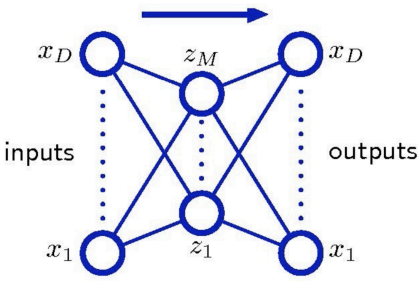
Motivation/Summary

Variational Autoencoders (VAEs) are powerful generative models, now having applications as diverse as from generating fake human faces, to producing purely synthetic music.

The plan for the first hour:

- We start by introducing **Autoencoders** as a nonlinear dimensionality reduction technique.
- We argue why they do not fit well the generative task.
- We discuss VAEs as a suitable alternative.
- VAEs incorporate two powerful techniques we learned recently: neural networks and variational inference.

Non-linear Dimension Reduction



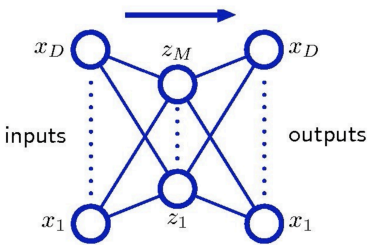
- Neural networks can be used for nonlinear dimensionality reduction.
- This is achieved by having the same number *of* outputs as inputs. These models are called autoencoders.
- Consider a feed-forward neural network that has D inputs, D outputs, and M hidden units, with $M < D$.
- We can squeeze the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

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

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

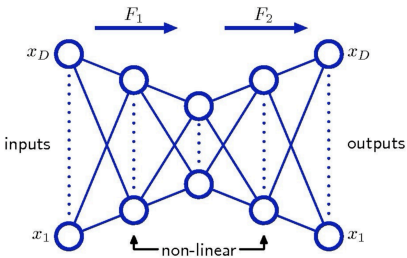
- We learn the parameters $\mathbf{w}$ by minimizing the reconstruction error:

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



- In the case when 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 (PCA).

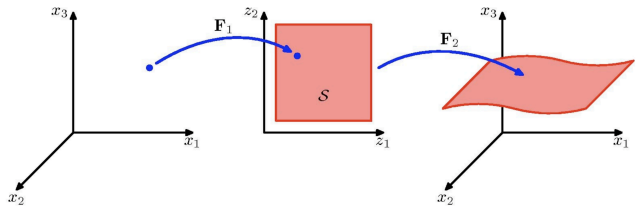
Deep Autoencoders



- 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.

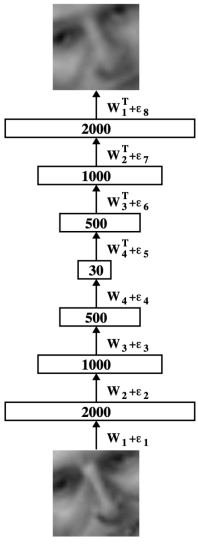
Geometrical Interpretation

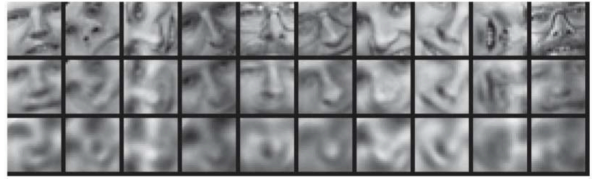
- Geometrical interpretation of the mappings performed by the network with 2 hidden layers for the case of $D = 3$ and $M = 2$ units in the middle layer.



- 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

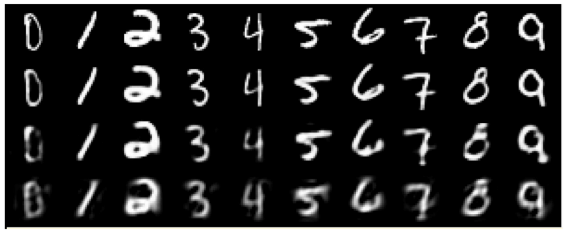




- We can consider very deep autoencoders.
- By row: Real data, Deep autoencoder with a bottleneck of 30 units, and 30-d PCA.

Deep Autoencoders

- Similar model for MNIST handwritten digits:



Real data

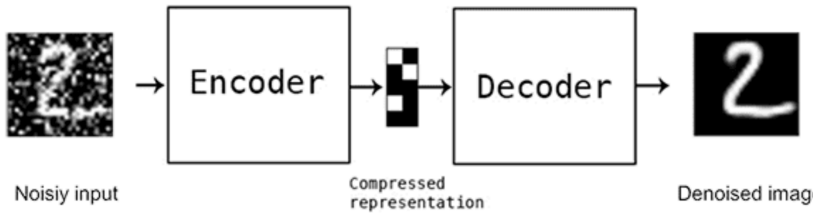
30-d deep autoencoder

30-d logistic PCA

30-d PCA

- 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).

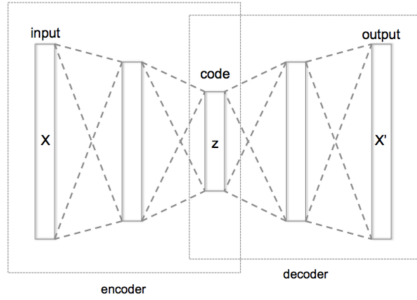
Autoencoders: Summary

Autoencoders reconstruct their input via an encoder and a decoder.

- **Encoder**: $e(x) = z \in F, \quad x \in X$
- **Decoder**: $d(z) = \hat{x} \in X$
- 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: $\hat{x} = d(e(x)) \approx x$.



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.

$$g(x_1) = z_1 \quad \Longrightarrow \quad f(z_1) = \hat{x}_1$$
$$g(x_2) = z_2 \quad \Longrightarrow \quad f(z_2) = \hat{x}_2$$

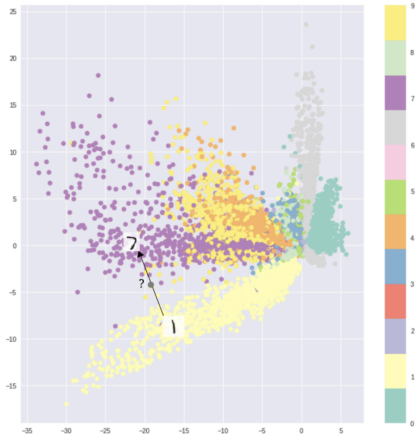
  - but proximity in feature space is not "directly" enforced for inputs in close proximity in data space, i.e.

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

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

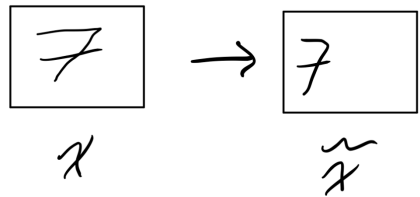
Issues with (deterministic) Autoencoders

- **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.



Issues with (deterministic) Autoencoders

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

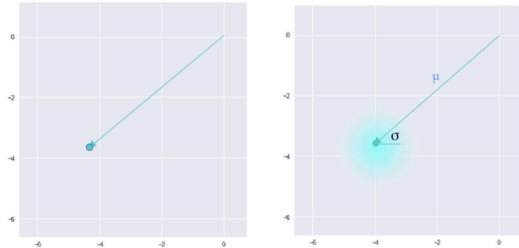


- The reconstruction looks quite good. However, if we chose a simple distance metric between inputs and reconstructions, we would heavily penalize the left-shift in the reconstruction $\tilde{x}$.
- Choosing an appropriate metric for evaluating model performance can be difficult, and that a miss-aligned 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 $p(z|x)$
- We assume that q_ϕ is a product of univariate normal distributions.
- Instead of the encoder learning an encoding vector, it learns two vectors: vector of means, μ, and another vector of standard deviations, σ.

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



Standard Autoencoder
(direct encoding coordinates)

Variational Autoencoder
(μ and σ initialize a probability distribution)

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

To minimize the reconstruction error, the algorithm may want to concentrate around the direct encodings. We will try to discourage this.

VAE: Specifics

- Our model is generated by the joint distribution over the latent codes and the input data $p_\theta(x, z)$:
  - The prior: $z \sim p_\theta(z)$ often Gaussian.
  - The likelihood: $x|z \sim \text{ExpFam}(x|d_\theta(z))$ with the decoder $d_\theta(z)$ given by a DNN,
    - e.g. for binary observations: $p_\theta(x|z) = \prod_{i=1}^{D} \text{Ber}(x_i|\sigma(d_\theta(z)))$
    - e.g. for continuous observations: $p_\theta(x|z) = \prod_{i=1}^{D} \mathcal{N}(x_i|d_\theta(z))$
- We could use the posterior $p_\theta(z|x) = p_\theta(x, z)/p_\theta(x)$ for encoding.

Issue: Learning $\mathbf{p(x) = \int p(x|z)p(z)dz}$ is intractable.
Introduce an approximation q_ϕ, with its own set of parameters ϕ, such that

$$q_\phi(z|x) \approx p_\theta(z|x).$$

VAE: Specifics

- Recall the idea behind Variational Inference:

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

  which is the (negative) loss function we use when training VAEs.
- First term in blue is the expected log-likelihood and the second is the divergence of q_ϕ from the prior.
- The encoder and decoder in a VAE become:
  - **Encoder**: $q_\phi(z|x)$, where
  $$q_\phi(z|x) = \mathcal{N}(\mu(x), \Sigma_\phi(x)),$$
  where $\mu_\phi(x)$ and diagonal $\Sigma_\phi(x)$ are computed by a deep NN
  - **Decoder**: $p(x|z) \sim \text{ExpFam}(x|d_\theta(z))$, where $d_\theta(z)$ is typically a deep neural network

VAE loss interpretation

The VAE maximization objective can be written as

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

- The first term handles the reconstruction loss.
- The second term regularizes the encoder not to be too far from the prior; this will make sure that the network cannot simply memorize the data.

β-VAE loss

$$\mathcal{L}(\theta, \phi|x) = \mathbb{E}_{z\sim q_\phi}\left[\log p_\theta(x|z)\right] - \beta\mathrm{KL}(q_\phi(z|x)||p(z)).$$

- $\beta > 1$: constrain latent bottleneck, enforce latent features to be uncorrelated.
- Produces features that are better disentangled.

Optimizing the ELBO

The ELBO for the whole dataset, scaled by its size N, is

$$\frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(\theta,\phi|x_n) = \frac{1}{N}\sum_{n=1}^{N}\left\{\mathbb{E}_{z\sim q_\phi(z|x_n)}\Big[\log p_\theta(x_n|z) + \log p_\theta(z) - \log q_\phi(z|x_n)\Big]\right\}$$

(note that ϕ is shared across samples = **amortized inference**)

- Optimize this with respect to θ and ϕ.
- We can create a mini-batch approximation of this objective.
- The gradient can be approximated using the reparametrization trick.

Let's recall this quickly.

Recall: Approximate the gradient for a single example

We use the reparametrization trick ($z \sim q_\phi(z|x_n)$ has the same distr. as $\mathcal{T}_\phi(\epsilon)$ with $\epsilon \sim p_0$)

$$\begin{aligned}\nabla_\phi \mathcal{L}(\theta, \phi|x_n) &= \nabla_\phi \mathbb{E}_{z\sim q_\phi(z|x_n)}\Big[\log p_\theta(x_n|z) + \log p_\theta(z) - \log q_\phi(z|x_n)\Big] \\ &= \mathbb{E}_{\epsilon\sim p_0(\epsilon)} \nabla_\phi \Big[\log p_\theta(x_n|\mathcal{T}_\phi(\epsilon)) + \log p_\theta(\mathcal{T}_\phi(\epsilon)) - \log q_\phi(\mathcal{T}_\phi(\epsilon)|x_n)\Big].\end{aligned}$$

In our case, q_ϕ is a product of normals with $\phi = (\mu_1, \ldots, \mu_M, \log\sigma_1, \ldots, \log\sigma_M)$

- The gradients of all terms with respect to ϕ are easy to compute.
- Since $\phi = \phi(x)$ is a NN, we can compute all gradients using backpropagation.

Similar idea for $\nabla_\theta \mathcal{L}(\theta, \phi|x_n)$. (here no reparametrization trick is needed).

Recall: Stochastic VI

Approximate $\frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(\theta,\phi|x_n)$ with $\mathcal{L}(\theta,\phi|x_n)$ for a single sample x_n from the dataset.

Get an approximate of the gradient $\nabla_\phi\mathcal{L}(\theta,\phi|x_n)$ using simple Monte Carlo

- Sample $z_i \sim q_\phi(z|x_n)$. (latent representations in our feature space)
- Compute the gradient of ELBO at each sample and take the average.

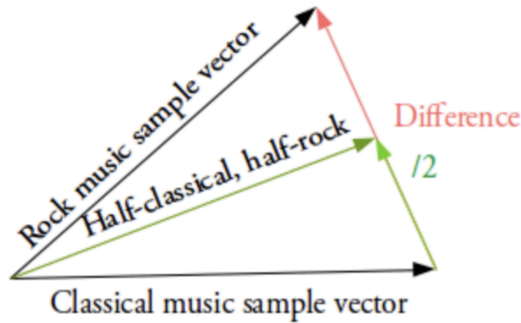
Update the parameters ϕ using the SGD step. Then do the same for θ.

After VAE is trained

- Once a VAE is trained, we can sample new inputs (generative model)

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

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



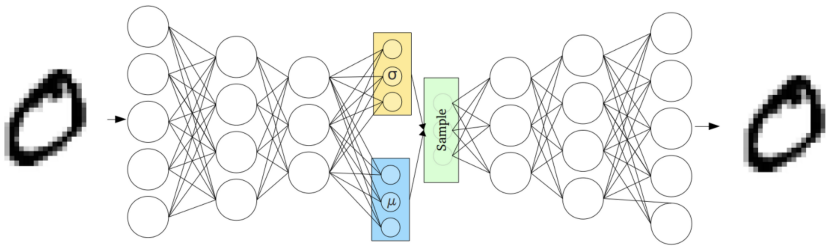
Interpolating between samples

Example: MNIST

- We choose the prior on z to be the standard Gaussian: $p(z) \sim \mathcal{N}(0, I)$.
- The likelihood function: $p_\theta(x|z) \sim \prod_{i=1}^{D} \text{Bernoulli}(\theta_i)$.
- Approximate posterior: $q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \Sigma_\phi(x))$, where Σ_ϕ diagonal.
- Finally, we use neural networks as our encoder and decoder
  - **Encoder**: $e_\phi(x) = [\mu_\phi(x), \log \Sigma_\phi(x)]$
  - **Decoder**: $d_\theta(z) = (\theta_1(z), \ldots, \theta_D(z))$
  - where θ_i are parameters of a Bernoulli rv for each input pixel.
- To get our reconstructed input, we simply take

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

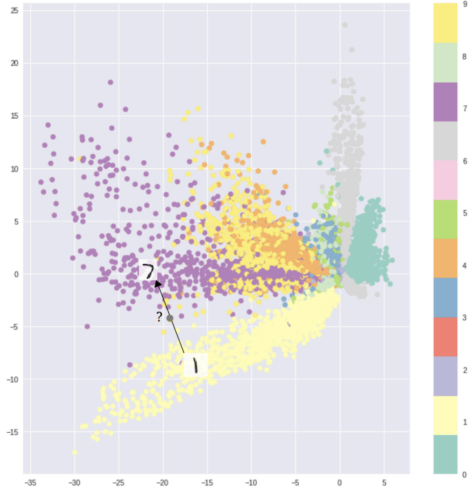
Example: MNIST

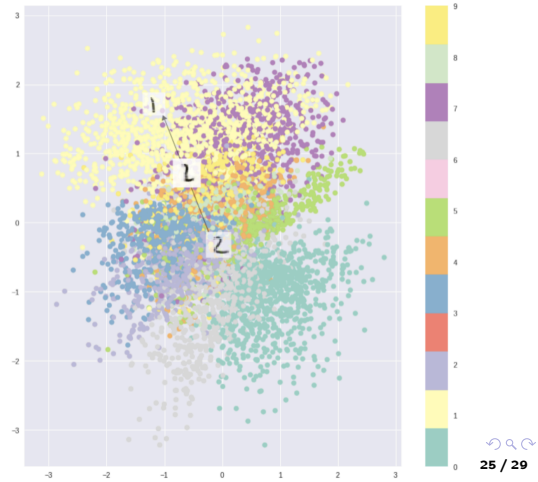


- We use neural networks for both the encoder and the decoder.
- We compute the loss function $\mathcal{L}(\theta, \phi; x)$ and propagate its derivative with respect to θ and ϕ, $\nabla_\theta \mathcal{L}$, $\nabla_\phi \mathcal{L}$, through the network during training.
- We use reparametrization trick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

MNIST: Autoencoder vs VAE

Codes generated by L: AE R: VAE





Extensions to VAEs: Disentangled Representations

- A large body of work attempts to improve the properties of the latent space by learning a **disentangled** representation.

- **Goal:** Each dimension of the latent space represents an independent real-world factor (e.g., uncovering head pose or hair color as independent factors in face images).

- Methods generally add regularization terms to either the posterior $q(\mathbf{z}|\mathbf{x})$ or the aggregated posterior $q(\mathbf{z}) = \frac{1}{J}\sum_{i=1}^{J} q(\mathbf{z}|\mathbf{x}_i)$.

  The new loss function becomes:

$$L_{new} = \text{ELBO}[\boldsymbol{\theta}, \boldsymbol{\phi}] - \lambda_1 \mathbb{E}_{Pr(\mathbf{x})}[\mathbf{R}_1[q(\mathbf{z}|\mathbf{x})]] - \lambda_2 \mathbf{R}_2[q(\mathbf{z})]$$

  where λ_1, λ_2 are weights, and $\mathbf{R}_1, \mathbf{R}_2$ are functions of the posterior and aggregated posterior.

Issues with VAEs

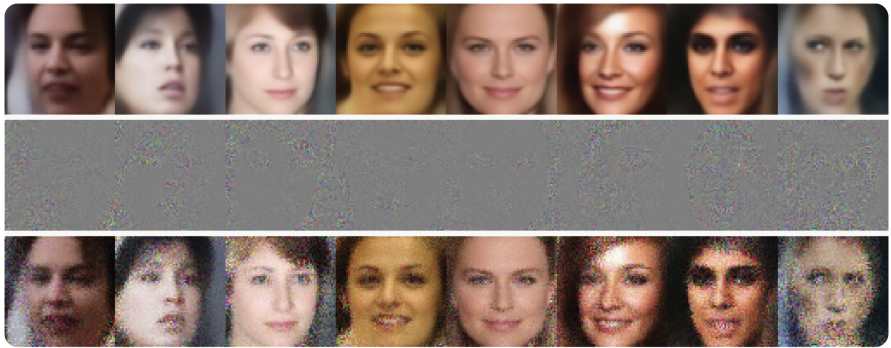
While powerful, VAEs have several notable drawbacks:

- **Intractable Likelihoods:** We cannot compute the likelihood of a new point $\mathbf{x}$ efficiently. It involves integrating over the hidden variable $\mathbf{z}$. Approximating this with Markov chain Monte Carlo (MCMC) is highly inefficient.

- **Posterior Collapse:** Training VAEs (especially sequence VAEs) can be brittle. The system may converge to a local minimum where the latent variable is completely ignored, and the encoder simply predicts the prior (see Tutorial).
  - **Solution:** Use an annealing schedule to only gradually introduce the KL divergence term in the cost function.

Issues with VAEs: Sample Quality

- Samples generated from VAEs are generally not perfect.
- The naive spherical Gaussian noise model (independent for each variable) typically produces unrealistic results.



Top row: Predicted means (overly smooth). **Middle row:** Instantiation of per-pixel spherical Gaussian noise. **Bottom row:** Final sample (sum of the first two rows, which appears overly noisy). Typical drawback of sampling images via VAEs.

Summary of Generative Models

The VAE is an architecture designed to learn a probability model over $Pr(\mathbf{x})$. While it excels at generating samples and interpolating between them via the latent variable $\mathbf{h}$, computing exact likelihoods remains difficult.

Some alternatives to VAEs:

- **Generative Adversarial Networks (GANs):**
  - Excellent for sampling, often producing sharper and more realistic images than VAEs.
  - Drawback: Like VAEs, they cannot evaluate the likelihood of new data points.

- **Normalizing Flows:**
  - A class of models where **both** sampling and exact likelihood evaluation are mathematically tractable.