Machine Learning

Lecture 11: Variational Autoencoder

Nevin L. Zhang

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology

This set of notes is based on internet resources and
Auto-encoding variational bayes DP Kingma, M Welling (2013). Auto-encoding variational bayes. `https://arxiv.org/abs/1312.6114`
C Doersch (2016). Tutorial on Variational Autoencoders.
`https://arxiv.org/abs/1606.05908`

Outline

Introduction

- So far, **supervised learning**
  - Discriminative methods:

    $$\{\mathbf{x}_i, y_i\}_{i=1}^N \rightarrow p(y|\mathbf{x})$$

  - Generative methods:

    $$\{\mathbf{x}_i, y_i\}_{i=1}^N \rightarrow P(y), p(\mathbf{x}|y)$$

- Next, **unsupervised learning**:
  - *Finite mixture models* for clustering [Skipped]

    $$\{\mathbf{x}_i\}_{i=1}^N \rightarrow P(z), p(\mathbf{x}|z)$$

  - *Varitional autoencoder* for data generation and representation learning

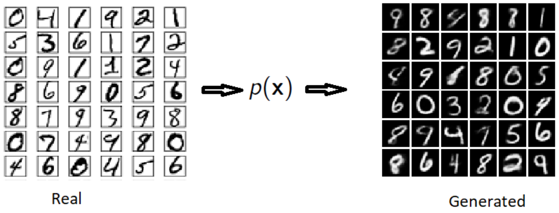
    $\{\mathbf{x}_i\}_{i=1}^N, p(\mathbf{z}) \rightarrow p(\mathbf{x}|\mathbf{z})$ $\quad q(\mathbf{z}|\mathbf{x})$ used in inference

  - *Generative adversarial networks* for data generation

    $$\{\mathbf{x}_i\}_{i=1}^N, p(\mathbf{z}) \rightarrow \mathbf{x} = g(\mathbf{z})$$

Outline

The Task



- Suppose we have an **unlabeled dataset** $\mathbf{X} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$, where each training example $\mathbf{x}^{(i)}$ is a vector that represents an image and each component of $\mathbf{x}^{(i)}$ represents a pixel in the image.
- We would like to learn a distribution $p(\mathbf{x})$ from the dataset so that we can generate more images that are similar (but different) to those in the dataset.
- If we can solve this task, then we have the ability to learn **very complex probabilistic model for high dimensional data**.
- The ability to generate realistic looking images would be useful for video game designers.

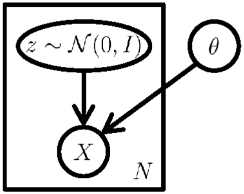
The Generative Model

- We assume that each image has a label $\mathbf{z}$ that is not observed. $\mathbf{z}$ is a vector of much lower dimension that $\mathbf{x}$.
- We further assume that the images are generated as follows:

$$\begin{aligned} \mathbf{z} &\sim p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I}) \text{ where } \mathbf{I} \text{ is the identity matrix} \\ \mathbf{x} &\sim p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) \text{ where } \boldsymbol{\theta} \text{ denotes model parameters} \end{aligned}$$

- Then we have

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) p(\mathbf{z}) d\mathbf{z}$$



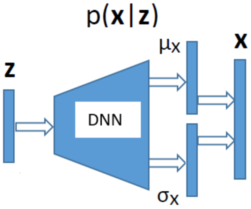
The Generative Model

- In addition, we assume that the conditional distribution is a Gaussian

$$p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\mu_x(\mathbf{z}, \boldsymbol{\theta}), \sigma_x^2(\mathbf{z}, \boldsymbol{\theta})\mathbf{I})$$

  - With mean vector is $\mu_x(\mathbf{z}, \boldsymbol{\theta})$ and diagonal covariance matrix is $\sigma_x(\mathbf{z}, \boldsymbol{\theta})\mathbf{I}$.
  - The mean vector $\mu_x(\mathbf{z}, \boldsymbol{\theta})$ and the vector $\sigma_x(\mathbf{z}, \boldsymbol{\theta})$ of sd's are deterministically determined by $\mathbf{z}$ via a **deep neural network** with parameters $\boldsymbol{\theta}$.

- So, we make use of the ability of neural network in representing complex functions to learn complicated probabilistic models.



Outline

The Likelihood Function

- To learn the model parameters, we need maximize the following likelihood function:

$$\log p_{\boldsymbol{\theta}}(\mathbf{X}) = \sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$$

where

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) = \log \int p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$$

- We want to use gradient ascent to maximize the likelihood function, which requires the gradient $\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$.
- The gradient is intractable because of the integration.

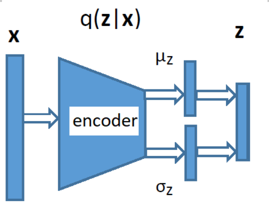
Naive Monte Carlo Gradient Estimator

- Here is a naive method to estimate $p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$ and hence the gradients.
- Sample L points $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(L)}$ from $p(\mathbf{z})$, and estimate $p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$ using

$$p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) \approx \frac{1}{L}\sum_{l=1}^{L} p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}^{(l)})$$

- Then we can compute $\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$.
- Unfortunately, this would not work.
  - The reason is that $\mathbf{x}$ is high-dimensional (thousands to millions of dimensions). Given $\mathbf{z}$, $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ is highly skewed, taking non-negligible values only in a very small region.
  - To state it another way, for a given data point $x^{(i)}$, $p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z})$ takes non-negligible values only for $\mathbf{z}$ from a very small region. As such, L needs to be extremely large for the estimate to be accurate.

Recognition Model



- To overcome the aforementioned difficulty, we introduce a **recognition model** $q_\phi(\mathbf{z}|\mathbf{x})$
$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\mu_z(\mathbf{x},\phi), \sigma_z^2(\mathbf{x},\phi)\mathbf{I})$$
- The mean vector $\mu_z(\mathbf{x},\phi)$ and the vector $\sigma_z(\mathbf{x},\phi)$ of sd's are deterministically determined by $\mathbf{z}$ via a **deep neural network** with parameters ϕ.
- We hope to get from $q_\phi(\mathbf{z}|\mathbf{x}^{(i)})$ samples of $\mathbf{z}$ for which $p_\theta(\mathbf{x}^{(i)}|\mathbf{z})$ has non-negligible values.
- The question now is: How to make use of $q_\phi(\mathbf{z}|\mathbf{x})$ when maximize the likelihood $\log p_\theta(\mathbf{X}) = \sum_{i=1}^N \log p_\theta(\mathbf{x}^{(i)})$.
- The answer is: **Variational inference**

The Variational Lower Bound

$$
\begin{aligned}
\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) &= E_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})\right] \\
&= E_{\mathbf{z}\sim q_\phi}\left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}^{(i)})}\right] \\
&= E_{\mathbf{z}\sim q_\phi}\left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}^{(i)})}\frac{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}\right] \\
&= E_{\mathbf{z}\sim q_\phi}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z})\right] - E_{\mathbf{z}\sim q_\phi}\left[\frac{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}{p_{\boldsymbol{\theta}}(\mathbf{z})}\right] + E_{\mathbf{z}\sim q_\phi}\left[\frac{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}^{(i)})}\right] \\
&= E_{\mathbf{z}\sim q_\phi}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z})\right] - \mathcal{D}_{KL}[q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z})] + E_{\mathbf{z}\sim q_\phi}\left[\frac{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}^{(i)})}\right. \\
&= \mathcal{L}(\mathbf{x}^{(i)},\boldsymbol{\theta},\phi) + \mathcal{D}_{KL}[q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}^{(i)})]
\end{aligned}
$$

So, we have the following variational lower bound on loglikelihood, which is tight if q has high capacity.

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) \geq \mathcal{L}(\mathbf{x}^{(i)},\boldsymbol{\theta},\phi)$$

The Variational Lower Bound: Alternative Perspective

$$E_{q(z|x)}\left[\log \frac{p(x,z)}{q(z|x)}\right]$$

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

$$\leq \log \int q(z|x) \frac{p(x,z)}{q(z|x)}\, dz$$

Jensen's inequality

$$= \log \int p(x,z)\, dz$$

$$= \log p(x)$$

$$E_{q(z|x)}\left[\log \frac{p(x,z)}{q(z|x)}\right]$$

$$= E_{q(z|x)}\left[\log \frac{p(z)\, p(x|z)}{q(z|x)}\right]$$

$$= E_{q(z|x)}[\log p(x|z)]$$

$$- E_{q(z|x)}\left[\log \frac{q(z|x)}{p(z)}\right]$$

$$\Downarrow$$

$$KL(q(z|x) \,||\, p(z))$$

Therefore:

$$\log p(x) \geq E_{q(z|x)}[\log (x|z)] - KL(q(z|x) \,||\, p(z))$$

The Objective Function

- Our new objective is to maximize the variational bound w.r.t both $\boldsymbol{\theta}$ and ϕ

$$\mathcal{L}(\mathbf{x}^{(i)}, \boldsymbol{\theta}, \phi) = E_{\mathbf{z} \sim q_{\phi}(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z})\right] - \mathcal{D}_{KL}[q_{\phi}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z})]$$

- Interpretation
  - The recognition model $q_{\phi}(\mathbf{z}|\mathbf{x}^{(i)})$ can be viewed as a **encoder** that takes a data point $\mathbf{x}^{(i)}$ and probabilistically encodes it into a latent vector $\mathbf{z}$.
  - The **decoder** $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ then takes the latent representation and probabilistically decodes it into a vector $\mathbf{x}$ in the data space.
  - The first term in $\mathcal{L}$ measure how well (the distribution of) the decoded output match the input $\mathbf{x}^{(i)}$. It is the **reconstruction error**.
  - The second term is a regularization terms that encourages the posterior distribution $q_{\phi}(\mathbf{z}|\mathbf{x}^{(i)})$ of the encoding $\mathbf{z}$ to be close to the prior $p_{\boldsymbol{\theta}}(\mathbf{z})$.
  - So, the method is called **variational autoencoder (VAE)**.

Illustration of Variational Autoencoder

$$\mathcal{L}(\mathbf{x}^{(i)}, \boldsymbol{\theta}, \phi) = E_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z})\right] - \mathcal{D}_{KL}[q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z})]$$

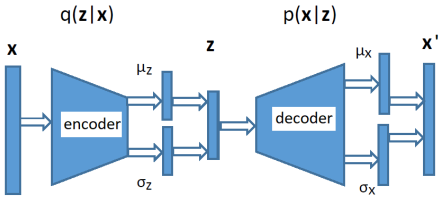
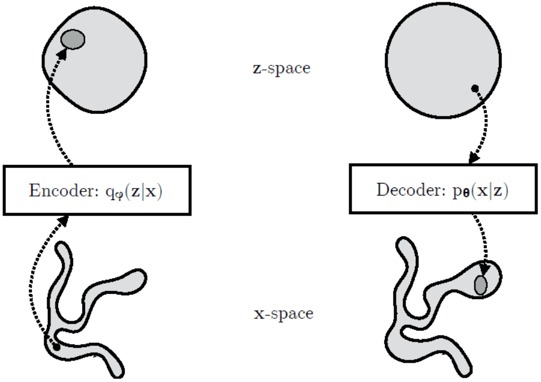
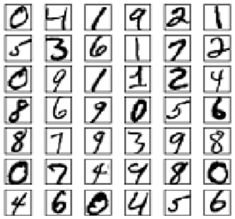


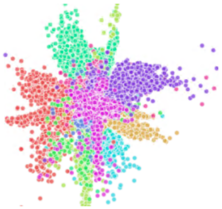
Illustration of Variational Autoencoder

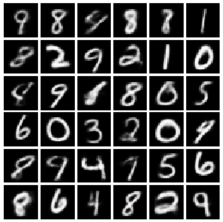

z-space
Encoder: $q_\varphi(\mathbf{z}|\mathbf{x})$
Decoder: $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$
x-space

- The encoder maps the data distribution, which is complex, to approximately an Gaussian distribution.
- The decoder maps a Gaussian distribution to the data distribution.

Illustration of Variational Autoencoder


Real

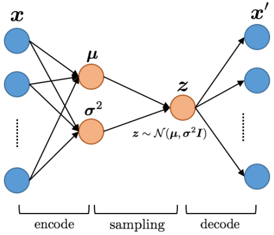

Latent Space


Fake

- Fake images generated by picking points in the latent space and map them back to the data space using the decoder.

Outline

The Need For Reparameterization



- The computation of the first term $\mathcal{L}_1$ of $\mathcal{L}$ requires **sampling**:

$$\mathcal{L}_1 = E_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}^{(i)})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}) \right] \approx \frac{1}{L} \sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)})$$

  where $\mathbf{z}^{(i,l)} \sim q_\phi(\mathbf{z}|\mathbf{x}^{(i)})$.

- But sampling looses gradient ∇_ϕ
  - While the LHS depends on ϕ, the RHS does not.
  - So, the stochastic connections from μ_z and σ_z to **z** makes backpropagation impossible.

The Reparameterization Trick

- Here is the **recognition model**

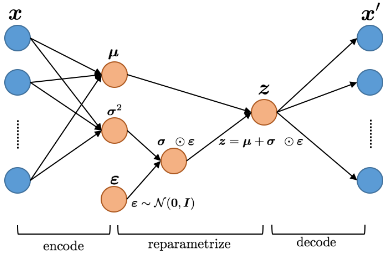
$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\mu_z(\mathbf{x}, \phi), \sigma_z^2(\mathbf{x}, \phi)\mathbf{I})$$

- Using the reparameterization trick, we change it into the following equivalent form

$$\mathbf{z} = \mu_z(\mathbf{x}, \phi) + \sigma_z(\mathbf{x}, \phi) \odot \boldsymbol{\epsilon}, \ \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

  where $\odot$ is element-wise product.

- Note that, now, $\mathbf{z}$ depends on μ_z, σ_z and $\boldsymbol{\epsilon}$ deterministically. $\boldsymbol{\epsilon}$ is stochastic, but it is an input the the network.


$\boldsymbol{x}$
$\boldsymbol{\mu}$
$\boldsymbol{\sigma}^2$
$\boldsymbol{\varepsilon}$
$\boldsymbol{\sigma} \odot \boldsymbol{\varepsilon}$
$\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\varepsilon}$
$\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$
$\boldsymbol{z}$
$\boldsymbol{x}'$
encode
reparametrize
decode

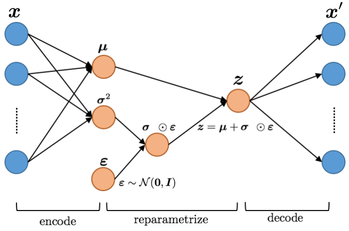
The Reparameterization Trick

- The reconstruction error term can be written as

$$\begin{aligned}\mathcal{L}_1 &= E_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z})\right] \\ &= E_{\boldsymbol{\epsilon}\sim p(\boldsymbol{\epsilon})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}(\mathbf{x}^{(i)},\phi,\boldsymbol{\epsilon}))\right] \approx \frac{1}{L}\sum_{l=1}^{L}\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)})\end{aligned}$$

  where $\mathbf{z}^{(i,l)} = \mathbf{z}(\mathbf{x}^{(i)},\phi,\boldsymbol{\epsilon}^{(l)}) = \mu_z(\mathbf{x}^{(i)},\phi) + \sigma_z(\mathbf{x}^{(i)},\boldsymbol{\phi}) \odot \boldsymbol{\epsilon}^{(l)}$, and $\boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$.

- Gradient $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\mathcal{L}_1$ can now be computed because, **for each given** ϵ, the network is deterministic.



x
μ
σ^2
ε
$\sigma \odot \varepsilon$
$\varepsilon \sim \mathcal{N}(0, I)$
z
$z = \mu + \sigma \odot \varepsilon$
x'
encode
reparametrize
decode

The Regularization Term

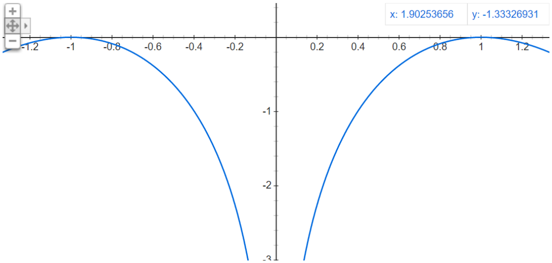
- The second term of $\mathcal{L}$ is $\mathcal{L}_2 = -\mathcal{D}_{KL}[q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z})]$.
- The two distributions involved are both Gaussian. Hence the term has a closed-form:

$$\mathcal{L}_2 = \frac{1}{2}\sum_{j=1}^{J}\left(1 + \log((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2\right)$$

  where log is natural logarithm, J is the dimension of $\mathbf{z}$, $\sigma_j^{(i)}$ is the j-th component of $\sigma_{\mathbf{z}}(\mathbf{x}^{(i)}, \phi)$, and $\mu_j^{(i)}$ is the j-th component of $\mu_{\mathbf{z}}(\mathbf{x}^{(i)}, \phi)$
- The gradient $\nabla_\phi \mathcal{L}_2$ is straightforward compute

The Regularization Term

“1+ln(x^2)-x^2”



The Final Objective Function

- Putting together, this is the objective function that we maximize using gradient ascent

$$\mathcal{L} \approx \frac{1}{L}\sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)}) + \frac{1}{2}\sum_{j=1}^{J}\left(1 + \log((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2\right)$$

  where $\mathbf{z}^{(i,l)} = \mu_z(\mathbf{x}^{(i)}, \phi) + \sigma_z^2(\mathbf{x}^{(i)}, \boldsymbol{\phi}) \odot \boldsymbol{\epsilon}^{(l)}$, and $\boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

- We have discussed how to compute the gradient $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\mathcal{L}$. Using the gradient, we can estimate $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ simultaneously using gradient ascent.

- The hyperparameter L is usually set to 1.

Comparison with Naive Monte Carlo

- Earlier, we mentioned a naive method for optimizing the parameters $\boldsymbol{\theta}$ of the generative model that involves the following objective:

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) \approx \log \frac{1}{L} \sum_{l=1}^{L} p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}^{(l)}) \tag{1}$$

  where the values of $\mathbf{z}$ are sampled from the prior $p(\mathbf{z})$. Those values do not depend on $\mathbf{x}^{(i)}$ and do not give high probabilities to $\mathbf{x}^{(i)}$. The RHS is a poor approximation of $\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$.

- Here is the our final objective function:

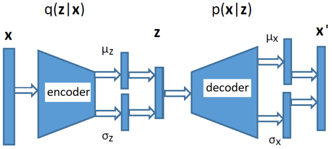
$$\mathcal{L} \approx \frac{1}{L} \sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)}) + \frac{1}{2} \sum_{j=1}^{J} \left(1 + \log((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2\right) \tag{2}$$

  where the values of $\mathbf{z}$ are sampled in such a way that they depend upon $\mathbf{x}^{(i)}$. Those values usually give high probabilities to $\mathbf{x}^{(i)}$. The LHS is a better approximation (lower bound) of $\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$.

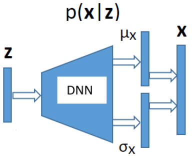
Outline

Example Generation

- During learning, both encoder and decoder are trained simultaneously.

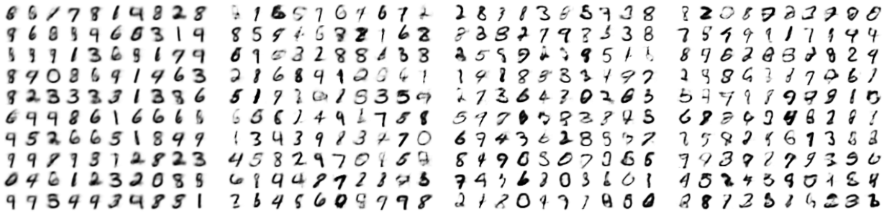


- To generate examples, we need on the decoder.
  - $\mathbf{z} \sim p(\mathbf{z})$, $\mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{z})$



Example Generation

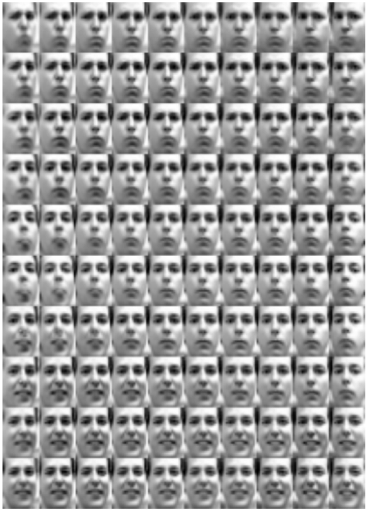
- One way to generate examples is to sample $\mathbf{z}$ from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ and then sample $\mathbf{x}$ from $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$.
- Here are images sampled from VAE's learned from the MNIST dataset. Note that several values are used for the dimensionality of $\mathbf{z}$.



(a) 2-D latent space (b) 5-D latent space (c) 10-D latent space (d) 20-D latent space

Example Generation

- Alternatively, we can manually pick $\mathbf{z}$ and sample $\mathbf{x}$ from $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$. This allows us to interpret each dimension of $\mathbf{z}$.
- Here are images sampled from VAE's learned from the Frey Face dataset. We can see that
  - X-axis represents head pose, and
  - Y-axis represent degree of smile.

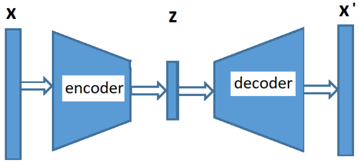


Outline

Discussions

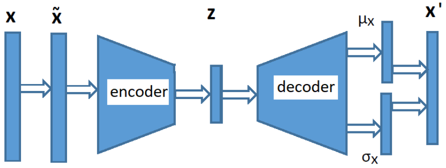
- The key reason for the excitement in VAE is that it shows that, by combining deep learning and the probabilistic approach, we can now learn complicated and high quality probabilistic models.
- In terms of specific functionality,
  - The decoder $p(\mathbf{x}|\mathbf{z})$ of VAE can be used to generate samples (images) that are similar (but different) to those in the training set.
  - The decoder gives a distribution $p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$, which can be approximated using the variational lower bound.
  - The encoder $q(\mathbf{z}|\mathbf{x})$ can be used to obtain low dimension latent representation of data.

Autoencoders



- While variational autoencoder is a probabilistic model, **autoencoder** is deterministic. The model parameters are trained by minimizing the reconstruction errors
$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||^2$$
- It is designed to learn a latent representation of data.
- However, it does not define a probability distribution $p(\mathbf{x})$ over data space and does not to generate new samples.
- Recent work on AE: He, Kaiming, et al. "Masked autoencoders are scalable vision learners." arXiv preprint arXiv:2111.06377 (2021).

Denoising Autoencoders

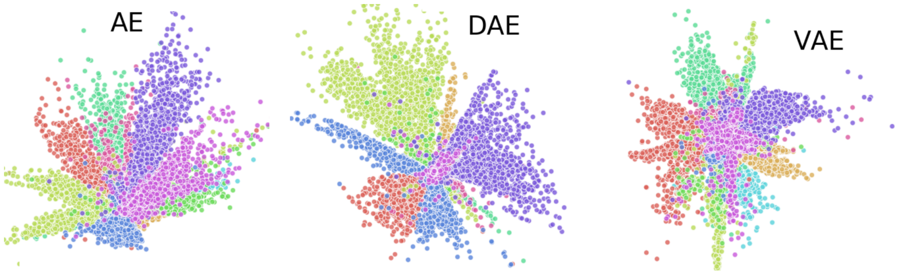


- Denoising autoencoder is a probabilistic model. The input $\mathbf{x}$ is randomly corrupted using $C(\tilde{\mathbf{x}}|x)$ to get $\tilde{\mathbf{x}}$ and the weights are optimized to minimize the following objective function:

$$-E_{\mathbf{x}\sim p_{data}} E_{\tilde{\mathbf{x}}\sim C(\tilde{\mathbf{x}}|x)} \log p(\mathbf{x}' = \mathbf{x}|\mathbf{z}(\tilde{\mathbf{x}}))$$

- It is more robust than autoencoder as a tool for learning latent representation of data.
- However, it does not define a probability distribution $p(\mathbf{x})$ over data space and does not to generate new samples.

Data Distribution in Latent Space (MNIST)



- VAE: Forces data into a normal distribution in the latent space.
- DAE: Preserves class separation better.

Recent Developments: Flow-Based Generative Models [1]

- Generative Model:
  - $\mathbf{z} \sim p_z(\mathbf{z})$ (Gaussian), $\mathbf{x} = g_\theta(\mathbf{z})$,
  - $g_\theta(\mathbf{z})$ is invertible, i.e., exists $f_\theta(\mathbf{x})$ such that $g_\theta(f_\theta(\mathbf{x})) = \mathbf{x}$.
  - Consequently, $\mathbf{z}$ has the same dimensionality as $\mathbf{x}$.
  - The model defines a distribution over inputs:

$$p_\theta(\mathbf{x}) = p_z(f_\theta(\mathbf{x}))|det(\frac{\partial f_\theta(\mathbf{x})}{\partial \mathbf{x}^\top})|$$

  - f_θ is implemented as a sequence of invertible functions, called **flows**, each represented as an CNN. See references.

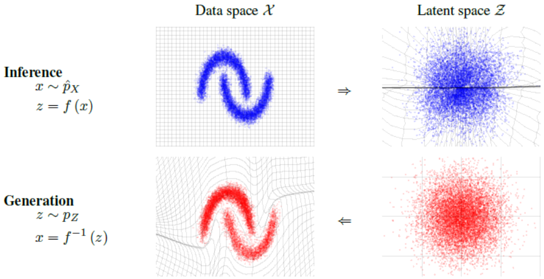
[1] Dinh L, Sohl-Dickstein J, Bengio S. Density estimation using Real NVP[J]. arXiv preprint arXiv:1605.08803, 2016; Kingma D P, Dhariwal P. Glow: Generative flow with invertible 1×1 convolutions[C]//Advances in Neural Information Processing Systems. 2018: 10236-10245.

Recent Developments: Flow-Based Generative Models

- Objective function for learning: maximizing the loglikelihood

$$\log p_\theta(\mathbf{X}) = \sum_{i=1}^{N} \log p_\theta(\mathbf{x}^{(i)}) = \sum_{i=1}^{N} \log p_z(f_\theta(\mathbf{x}^{i})) + \sum_{i=1}^{N} \log |det(\frac{\partial f_\theta(\mathbf{x}^{(i)})}{\partial (\mathbf{x}^{(i)})^\top})|$$

- Intuitively, pick θ so that the CNN map images from their original space, where they are not Gaussian distributed, to a latent space where they are Gaussian distributed.



Recent Developments: Flow-Based Generative Models

- Image Synthesis: $\mathbf{z} \sim p_z(\mathbf{z})$ (Gaussian), $\mathbf{x} = g_\theta(\mathbf{z})$