Chapter 14

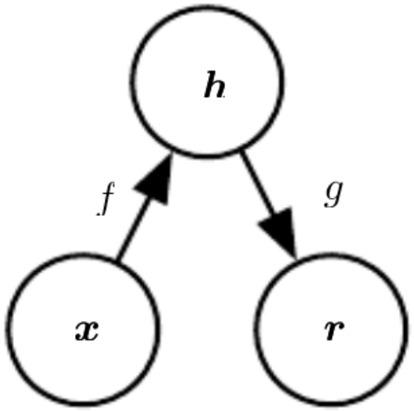
Autoencoders

Autoencoders

- A type of neural networks trained to copy approximately its input to its output in the hopes of learning useful features
- The network of an autoencoder may be viewed as containing an encoder and a decoder, specifying deterministic or stochastic mappings

$$\text{Encoder:} \quad \boldsymbol{h} = f(\boldsymbol{x}) \text{ or } p_{\text{model}}(\boldsymbol{h}|\boldsymbol{x})$$
$$\text{Decoder:} \quad \boldsymbol{r} = g(\boldsymbol{h}) \text{ or } p_{\text{model}}(\boldsymbol{x}|\boldsymbol{h})$$

where the hidden layer $\boldsymbol{h}$ describes a code used to represent $\boldsymbol{x}$



- The learning is to minimize a loss function, likely with regularization

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) + \Omega(\boldsymbol{h}, \boldsymbol{x})$$

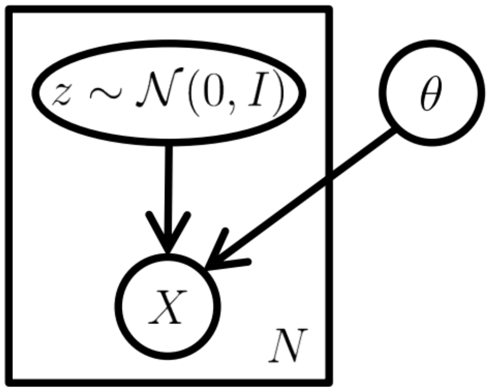
- Most learning techniques for training feedforward networks can apply
- Traditionally, autorencoders were used for dimension reduction
- However, theoretical connections between autoencoders and some modern latent variable models have brought autoencoders to the forefront of generative modeling

Variational Autoencoders (VAE)

- A probabilistic generative model with latent variables that is built on top of end-to-end trainable neural networks

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; \mathbf{0}, \boldsymbol{I})$$

$$p(\boldsymbol{x}|\boldsymbol{z}) = \underbrace{p(\boldsymbol{x}; o(\boldsymbol{z}; \boldsymbol{\theta}))}_{\text{Neural Networks}} = \mathcal{N}(\boldsymbol{x}; o(\boldsymbol{z}; \boldsymbol{\theta}), \sigma^2 \boldsymbol{I})$$



Training VAE

- To determine $\boldsymbol{\theta}$, we would intuitively hope to maximize the marginal distribution $p(\boldsymbol{x};\boldsymbol{\theta})$

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

- This however becomes difficult as the integration over $\boldsymbol{z}$ is in general intractable when $p(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})$ is modeled by a neural network

- To circumvent this difficulty, we recall that

$$\log p(\boldsymbol{X};\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{X}, q, \boldsymbol{\theta}) + \mathsf{KL}(q(\boldsymbol{Z})||p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}))$$

where

$$\mathcal{L}(\boldsymbol{X}, q, \boldsymbol{\theta}) = \int q(\boldsymbol{Z})\log p(\boldsymbol{X},\boldsymbol{Z};\boldsymbol{\theta})d\boldsymbol{Z} - \int q(\boldsymbol{Z})\log q(\boldsymbol{Z})d\boldsymbol{Z}$$

$$\mathsf{KL}(q(\boldsymbol{Z})||p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta})) = \int q(\boldsymbol{Z})\log\frac{q(\boldsymbol{Z})}{p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta})}d\boldsymbol{Z}$$

- A rearrangement gives

$$\log p(\boldsymbol{X};\boldsymbol{\theta}) - \mathsf{KL}(q(\boldsymbol{Z})||p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta})) = \mathcal{L}(\boldsymbol{X}, q, \boldsymbol{\theta})$$

- As the equality holds for any choice of $q(\boldsymbol{Z})$, we introduce a distribution $q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')$ modeled by another neural network with parameter $\boldsymbol{\theta}'$ to obtain

$$\log p(\boldsymbol{X};\boldsymbol{\theta}) - \mathsf{KL}(q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')||p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta})) = \mathcal{L}(\boldsymbol{X}, q, \boldsymbol{\theta})$$

- The right hand side can be spell out as

$$\begin{aligned}
\mathcal{L}(\boldsymbol{X}, q, \boldsymbol{\theta}) = \quad & E_{\boldsymbol{Z}\sim q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')} \log p(\boldsymbol{X}|\boldsymbol{Z};\boldsymbol{\theta}) \\
& + E_{\boldsymbol{Z}\sim q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')} \log p(\boldsymbol{Z}) - E_{\boldsymbol{Z}\sim q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')} \log q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}') \\
= \quad & E_{\boldsymbol{Z}\sim q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')} \log p(\boldsymbol{X}|\boldsymbol{Z};\boldsymbol{\theta}) \\
& - \mathsf{KL}(q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')||p(\boldsymbol{Z}))
\end{aligned}$$

- Now, instead of directly maximizing the intractable $p(\boldsymbol{X};\boldsymbol{\theta})$, we attempt to maximize

$$\log p(\boldsymbol{X};\boldsymbol{\theta}) - \mathsf{KL}(q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')||p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}))$$

which amounts to maximizing

$$\underbrace{E_{\boldsymbol{Z}\sim q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')} \log p(\boldsymbol{X}|\boldsymbol{Z};\boldsymbol{\theta})}_{\text{Reconstruction}} \underbrace{-\mathsf{KL}(q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')||p(\boldsymbol{Z}))}_{\text{Regularization}}$$

- To make the KL divergence tractable, both $q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')$ and $p(\boldsymbol{Z})$ are assumed to be Gaussians

- A by-product of this training process is a stochastic encoder

$$q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}') \approx p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta})$$

- The reconstruction term requires that the latent code $\boldsymbol{Z}$ generated by the encoder $q(\boldsymbol{Z}|\boldsymbol{X};\theta')$ for the input $\boldsymbol{X}$ should maximize the log-likelihood $\log p(\boldsymbol{X}|\boldsymbol{Z};\boldsymbol{\theta})$ of $\boldsymbol{X}$
- The regularization term requires that the conditional distribution $q(\boldsymbol{Z}|\boldsymbol{X};\theta')$ of the latent code $\boldsymbol{Z}$ given $\boldsymbol{X}$ should be compatible with the prior $p(\boldsymbol{Z})$
- Even though the reconstruction term can be evaluated by sampling $\boldsymbol{Z}$ from $q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')$, it becomes difficult to compute the gradient w.r.t. $\boldsymbol{\theta}'$

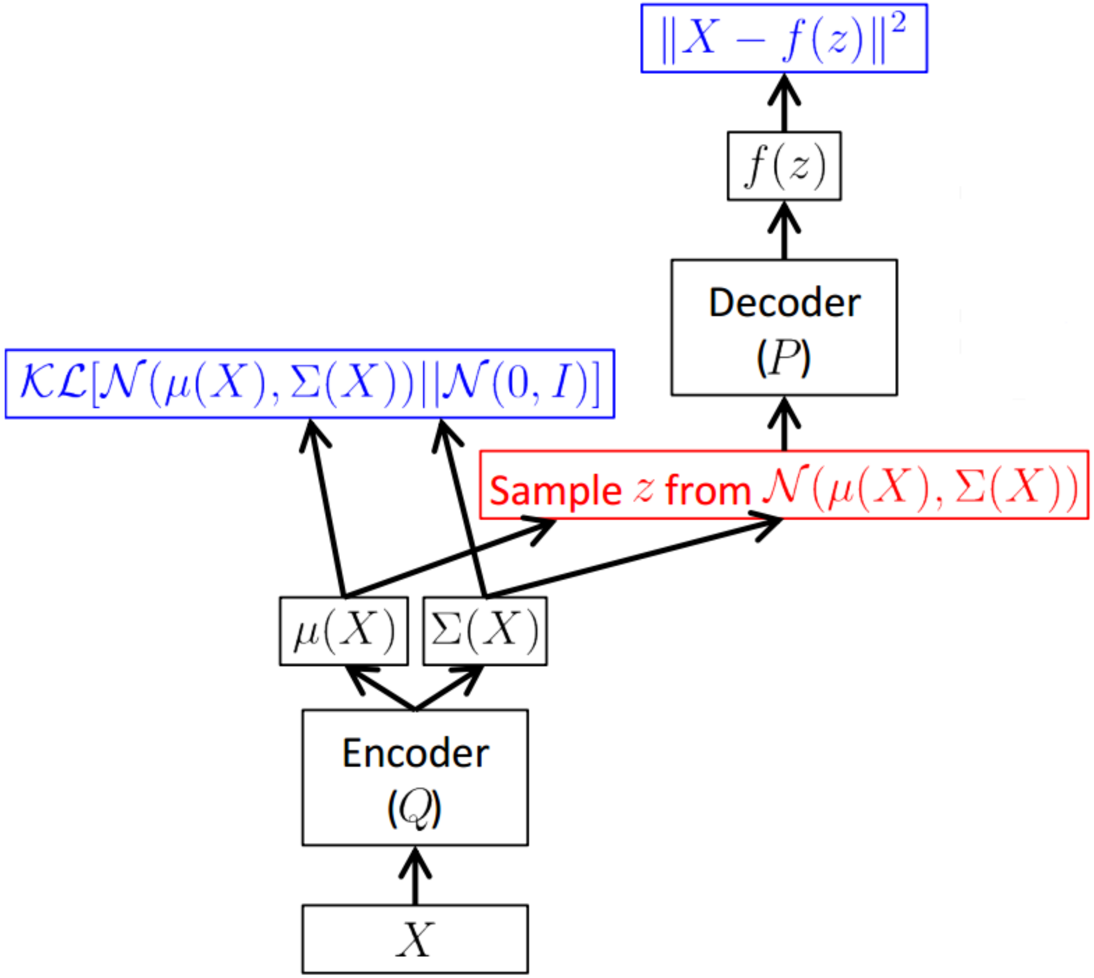
$$E_{\boldsymbol{Z}\sim q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')} \log p(\boldsymbol{X}|\boldsymbol{Z};\boldsymbol{\theta})$$

- The re-parameterization technique works around this difficulty by generating samples input to the decoder with

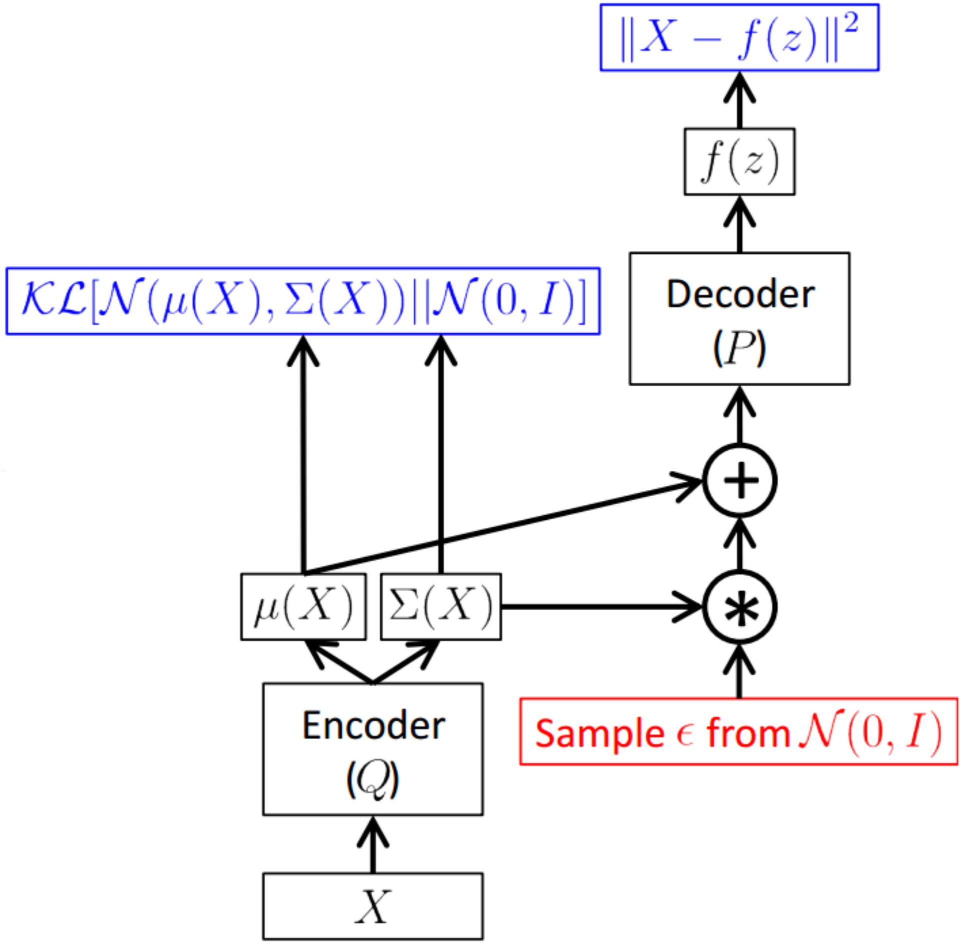
$$\boldsymbol{B}(\boldsymbol{X})\boldsymbol{\epsilon} + \boldsymbol{\mu}(\boldsymbol{X})$$

where $\boldsymbol{B}\boldsymbol{B}^T = \Sigma$ and $\epsilon \sim \mathcal{N}(0, \boldsymbol{I})$

- In fact, the encoder can learn $\boldsymbol{B}(\boldsymbol{X})$ directly



$$\underbrace{E_{\boldsymbol{Z}\sim q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')} \log p(\boldsymbol{X}|\boldsymbol{Z};\boldsymbol{\theta})}_{\text{Sampling needed}} - \mathrm{KL}(q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')||p(\boldsymbol{Z}))$$



$$\underbrace{E_{\boldsymbol{Z} \sim q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')} \log p(\boldsymbol{X}|\boldsymbol{Z};\boldsymbol{\theta})}_{\text{Re-parameterization for end-to-end training}} \quad - \mathrm{KL}(q(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}')||p(\boldsymbol{Z}))$$

- Given the data $\boldsymbol{X} = \{\boldsymbol{x}_i\}$ is drawn from an empirical distribution $p_d(\boldsymbol{x})$, the objective function $\mathcal{L}(\boldsymbol{X}, q, \boldsymbol{\theta})$ can be expressed more precisely as

$$\frac{1}{N}\sum_{i=1}^{N}\left(E_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}')}\log p(\boldsymbol{x}^{(i)}|\boldsymbol{z};\boldsymbol{\theta}) - \mathsf{KL}(q(\boldsymbol{z}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}')||p(\boldsymbol{z}))\right)$$

- It is convenient to write

$$E_{\boldsymbol{x}\sim p_d(\boldsymbol{x})}[E_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta}')}\log p(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})] - \underbrace{E_{\boldsymbol{x}\sim p_d(\boldsymbol{x})}[\mathsf{KL}(q(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta}')||p(\boldsymbol{z}))]}_{\text{Regularization}}$$

- Further insights into the regularization term can be gained by rewriting the regularization term

$$E_{\boldsymbol{x}\sim p_d(\boldsymbol{x})}[E_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta}')}\log p(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})] + \underbrace{E_{\boldsymbol{x}\sim p_d(\boldsymbol{x})}[H(q(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta}'))]}$$
$$- \underbrace{E_{\boldsymbol{z}\sim q(\boldsymbol{z})}[-\log p(\boldsymbol{z})]}_{\text{Cross Entropy}}$$

where

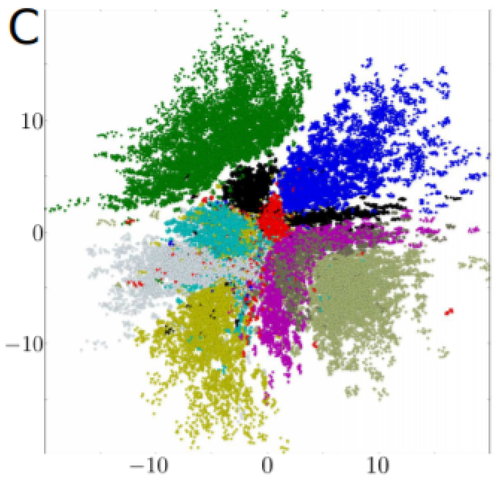
- $H(q(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta}'))$ is the conditional entropy of $\boldsymbol{z}$ at encoder output
- $q(\boldsymbol{z}) = \int p_d(\boldsymbol{x})q(\boldsymbol{z}|\boldsymbol{x})d\boldsymbol{x}$ is the aggregated distribution of $\boldsymbol{z}$

- Likewise, it can be reformulated as

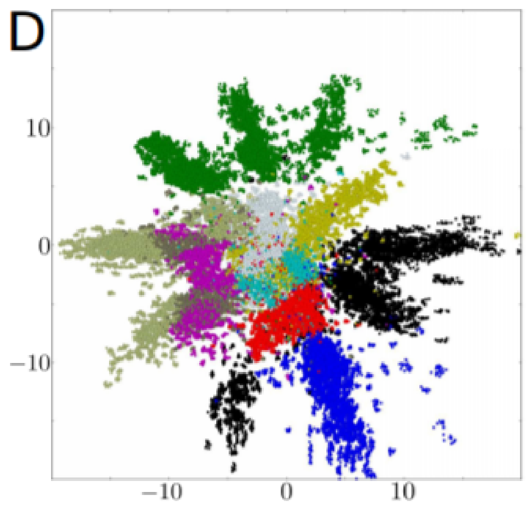
$$E_{\boldsymbol{x}\sim p_d(\boldsymbol{x})}[E_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta}')}\log p(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})] - \underbrace{(H(\boldsymbol{x}) - E_{\boldsymbol{z}\sim q(\boldsymbol{z})}[H(q(\boldsymbol{x}|\boldsymbol{z}))])}_{\text{Mutual information between } \boldsymbol{x} \text{ and } \boldsymbol{z}} - \underbrace{\mathsf{KL}(q(\boldsymbol{z})||p(\boldsymbol{z}))}_{\text{KL div. between the aggregated and prior dist.}}$$

- When the encoder is viewed as a communication channel with $\boldsymbol{x}$ as input and $\boldsymbol{z}$ as output, the mutual information indicates how much information about $\boldsymbol{x}$ is sent to the $\boldsymbol{z}$; the larger the mutual information, the more information about $\boldsymbol{x}$ the $\boldsymbol{z}$ carries

- The training criterion encourages the conditional entropy to be large (i.e., the codes $\boldsymbol{z}$ for an input $\boldsymbol{x}$ to be diverse), or equivalently the mutual information to be low, and the aggregated distribution $q(\boldsymbol{z})$ to approximate the prior $p(\boldsymbol{z})$



(a) Gaussian prior



(b) GMM prior

Aggregated distributions on MNIST

https://arxiv.org/abs/1511.05644 (Adversarial Autoencoders)

Conditional VAE (CVAE)

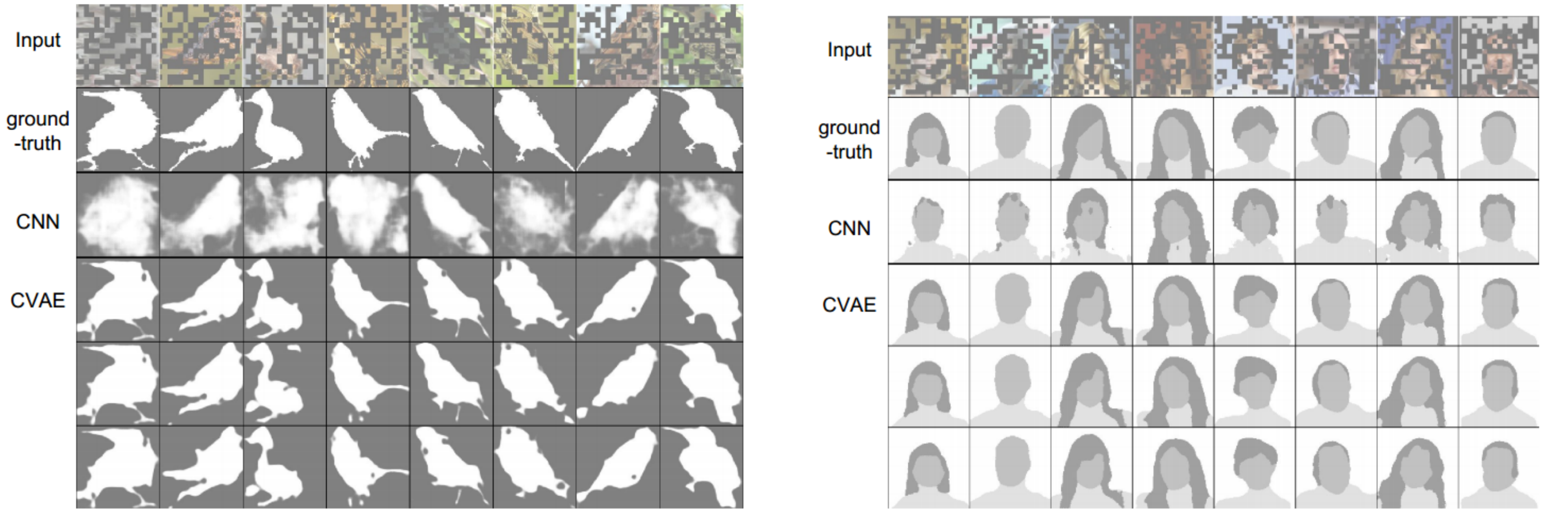
- **Idea:** Training VAE to learn a conditional distribution $p(\boldsymbol{X}|c)$
- Following the same line of derivations as for the unconditional case, the variational lower bound of $\log p(\boldsymbol{X}|c)$ for CVAE is given by

$$E_{\boldsymbol{Z}\sim q(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta}')} \log p(\boldsymbol{X}|\boldsymbol{Z},c;\boldsymbol{\theta}) - \mathsf{KL}(q(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta}')||\underbrace{p(\boldsymbol{Z}|c)})$$

- Now both the encoder $q(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta}')$ and the decoder $p(\boldsymbol{X}|\boldsymbol{Z},c;\boldsymbol{\theta})$ need to take c as part of their input
- How to specify the conditional prior $p(\boldsymbol{Z}|c)$?
  - Learn from data using a neural network (regularization?)
  - Use a simple fixed prior without regard to c
  - Ignore the regularization term (no longer VAE)

- At test time, samples can be generated by first drawing $\boldsymbol{Z} \sim p(\boldsymbol{Z}|c)$ and then passing it through the decoder $p(\boldsymbol{X}|\boldsymbol{Z}, c; \boldsymbol{\theta})$

- Learning structured outputs $\boldsymbol{X}$ based on corrupted inputs c



https://papers.nips.cc/paper/5775-learning-structured-output-representation-using-deep-conditional-generative-models

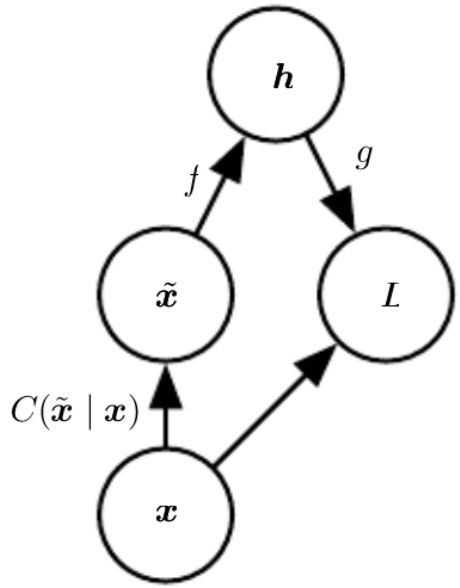
- At training time, the input image c is corrupted with part of its contents blocked randomly at different positions, and the conditional prior $p(\boldsymbol{Z}|c)$ is learned

Denoising Autoencoders (DAE)

- The DAE is to receive a corrupted data point as input and to predict the uncorrupted data point as output; that is, to minimize

$$L(\boldsymbol{x}, g(f(\tilde{\boldsymbol{x}})))$$

where $\tilde{\boldsymbol{x}}$ is a noise-corrupted version of $\boldsymbol{x}$



- To be precise, the training of DAE proceeds as follows
  1. Sample an $\boldsymbol{x}$ from the training data
  2. Sample a corrupted version $\tilde{\boldsymbol{x}}$ from $C(\tilde{\boldsymbol{x}}|\boldsymbol{x})$
  3. Minimize the negative log-likelihood by performing gradient descent w.r.t. model parameters

$$-\log p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h} = f(\tilde{\boldsymbol{x}}))$$

- When the encoder f is deterministic, the training of DAE is no different than training a feedforward network

- It is shown that when both $p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$ and $C(\tilde{\boldsymbol{x}}|\boldsymbol{x})$ are assumed to be Gaussian, i.e., training with

$$\min \|g(f(\tilde{\boldsymbol{x}})) - \boldsymbol{x}\|^2 \text{ and } C(\tilde{\boldsymbol{x}}|\boldsymbol{x}) \sim \mathcal{N}(\tilde{\boldsymbol{x}}; \boldsymbol{x}, \sigma^2 \boldsymbol{I}),$$

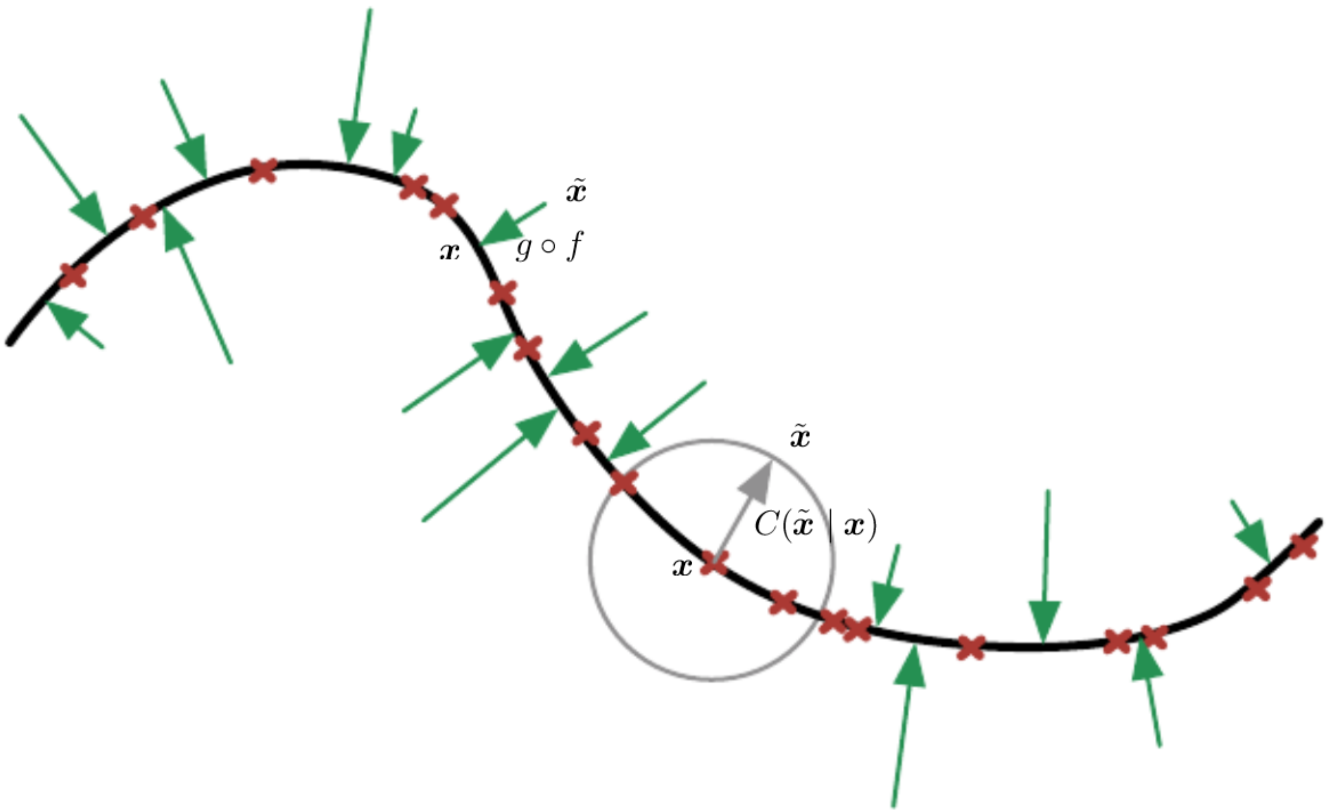
  the DAE learns a vector field $(g(f(\boldsymbol{x})) - \boldsymbol{x})$ that gives estimates of the score of the data distribution defined as

$$\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$$

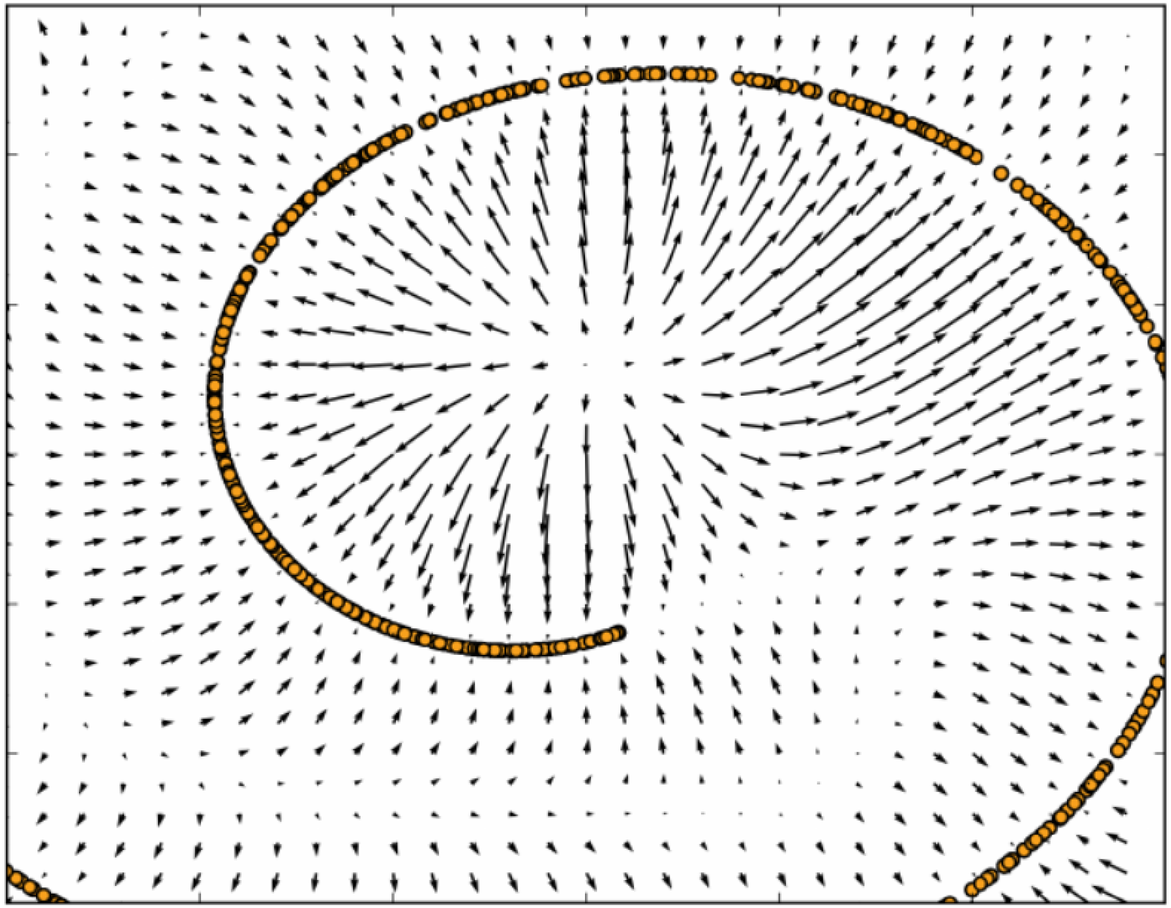
- Note that when $\|g(f(\tilde{\boldsymbol{x}})) - \boldsymbol{x}\|^2$ is minimized, we have

$$g(f(\tilde{\boldsymbol{x}})) \approx E_{\boldsymbol{x},\tilde{\boldsymbol{x}} \sim \hat{p}_{\text{data}}(\boldsymbol{x}) C(\tilde{\boldsymbol{x}}|\boldsymbol{x})}[\boldsymbol{x}|\tilde{\boldsymbol{x}}]$$

- Thus, $(g(f(\tilde{\boldsymbol{x}})) - \tilde{\boldsymbol{x}})$ is a vector that points approximately back to the nearest point on the data manifold



Green arrows: $g(f(\tilde{\boldsymbol{x}})) - \tilde{\boldsymbol{x}}$



Vector filed learned by a DAE

(Vector field has zeros at both maxima and minima of $p(\boldsymbol{x})$)

Sparse Autoencoders

- A sparse autoencoder is an autoencoder whose training criterion involves a sparsity penalty $\Omega(\boldsymbol{h})$

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) + \Omega(\boldsymbol{h})$$

- It can be interpreted as approximating the maximum likelihood training of a generative model $p_{\text{model}}(\boldsymbol{x}, \boldsymbol{h})$ with latent variables $\boldsymbol{h}$

$$\begin{aligned}\log p_{\text{model}}(\boldsymbol{x}) &= \log \sum_{\boldsymbol{h}} p_{\text{model}}(\boldsymbol{x}, \boldsymbol{h}) \\ &\approx \underbrace{\log p_{\text{model}}(\boldsymbol{h})}_{\Omega} + \underbrace{\log p_{\text{model}}(\boldsymbol{x}|\boldsymbol{h})}_{L},\end{aligned}$$

where the $p_{\text{model}}(\boldsymbol{h})$ is factorial and follows the Laplace prior

$$p_{\text{model}}(\boldsymbol{h}) = \frac{\lambda}{2} e^{-\lambda|h_i|}$$

Contractive Autoencoders (CAE)

- The CAE imposes a regularizer on the code $\boldsymbol{h}$ which encourages to learn an encoder function that does not change much when input $\boldsymbol{x}$ changes slightly

$$L(\boldsymbol{x}, g(f(\boldsymbol{x})) + \Omega(\boldsymbol{h}, \boldsymbol{x})$$

where

$$\Omega(\boldsymbol{h}, \boldsymbol{x}) = \lambda \left\| \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \right\|_F^2$$

- The encoder $f(\boldsymbol{x})$ at a training point $\boldsymbol{x}_0$ can be approximated as

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}_0) + \frac{\partial f(\boldsymbol{x}_0)}{\partial \boldsymbol{x}}(\boldsymbol{x} - \boldsymbol{x}_0)$$

- As such, the CAE is seen to encourage the Jacobian matrix $\partial f(\boldsymbol{x}_0)/\partial \boldsymbol{x}$ at every training point $\boldsymbol{x}_0$ to become contractive, making their singular values become as small as possible

- It is however noticed that the optimization has to respect also the reconstruction error; this leads to an effect that keeps the singular values along directions with large local variances
- These directions are known as tangent directions to the data manifold; that is, they correspond to real variations in the data
- The encoder learns a mapping $f(\boldsymbol{x})$ that is only sensitive to changes along the manifold directions

Review

- Stochastic vs. deterministic autoencoders
- Autoencoders vs. generative models with latent variables
- Training autoencoders vs. learning data manifolds