

Chapter 14

Autoencoders

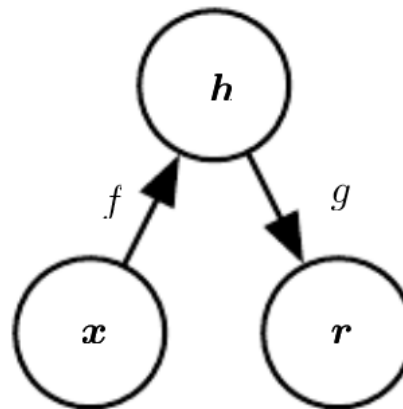
Autoencoders

- 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

Encoder: $\mathbf{h} = f(\mathbf{x})$ or $p_{\text{model}}(\mathbf{h}|\mathbf{x})$

Decoder: $\mathbf{r} = g(\mathbf{h})$ or $p_{\text{model}}(\mathbf{x}|\mathbf{h})$

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



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

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

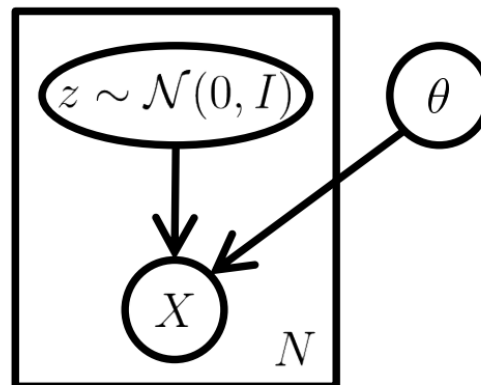
- Most learning techniques for training feedforward networks can apply
- Traditionally, autoencoders were used for dimension reduction
- However, theoretical connections between autoencoders and some modern latent variable models have brought autoencoders to the forefront of generative modeling

Variational Autoencoders (VAE)

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

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

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



Training VAE

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

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

- This however becomes difficult as the integration over \mathbf{z} is in general intractable when $p(\mathbf{x}|\mathbf{z}; \theta)$ is modeled by a neural network
- To circumvent this difficulty, we recall that

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

where

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

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

- A rearrangement gives

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

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

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

- The right hand side can be spell out as

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

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

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

which amounts to maximizing

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

- To make the KL divergence tractable, both $q(\mathbf{Z}|\mathbf{X}; \boldsymbol{\theta}')$ and $p(\mathbf{Z})$ are assumed to be Gaussians
- A by-product of this training process is a stochastic encoder

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

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

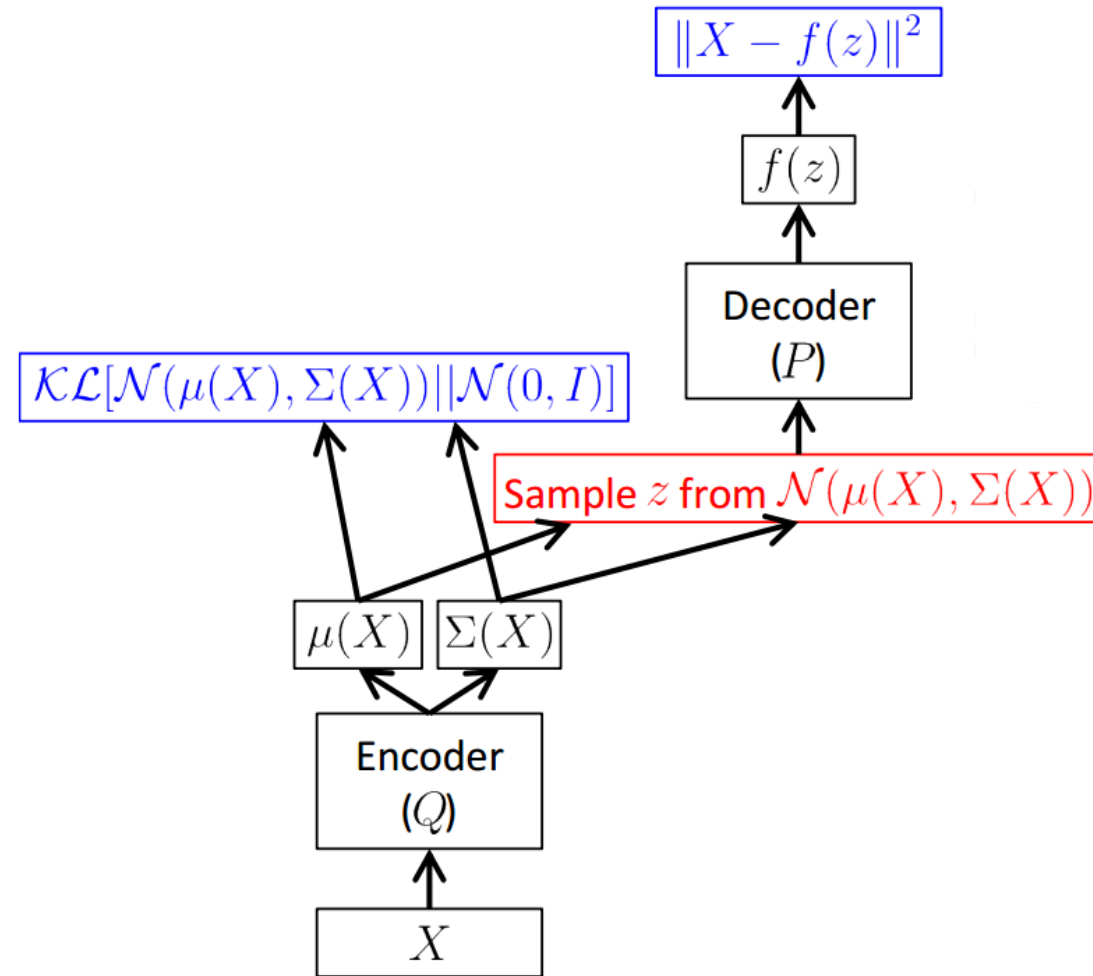
$$E_{\mathbf{Z} \sim q(\mathbf{Z}|\mathbf{X}; \theta')} \log p(\mathbf{X}|\mathbf{Z}; \theta)$$

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

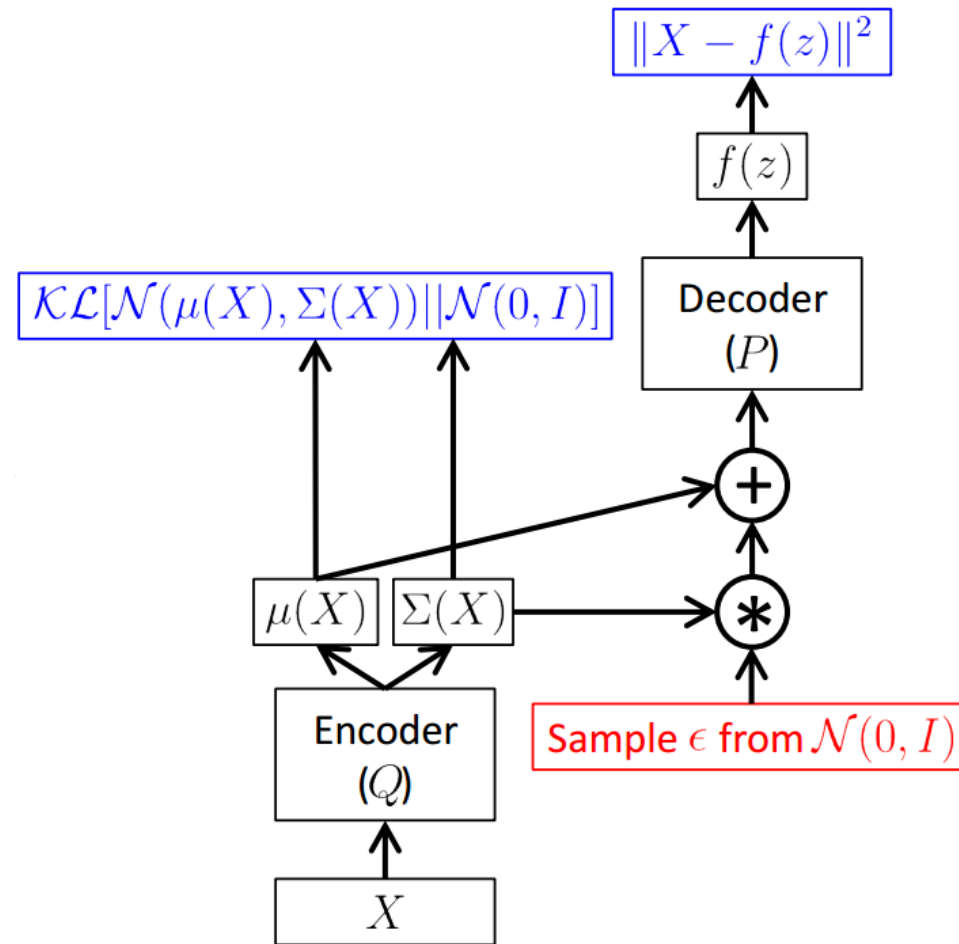
$$\mathbf{B}(\mathbf{X})\epsilon + \mu(\mathbf{X})$$

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

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



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



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

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

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

- It is convenient to write

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

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

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

where

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

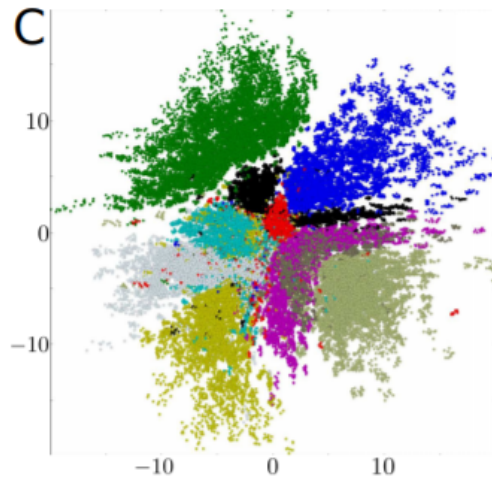
- Likewise, it can be reformulated as

$$E_{\mathbf{x} \sim p_d(\mathbf{x})} [E_{\mathbf{z} \sim q(\mathbf{z}|\mathbf{x}; \theta')} \log p(\mathbf{x}|\mathbf{z}; \theta)] - \underbrace{(H(\mathbf{x}) - E_{\mathbf{z} \sim q(\mathbf{z})} [H(q(\mathbf{x}|\mathbf{z}))])}_{\text{Mutual information between } \mathbf{x} \text{ and } \mathbf{z}}$$

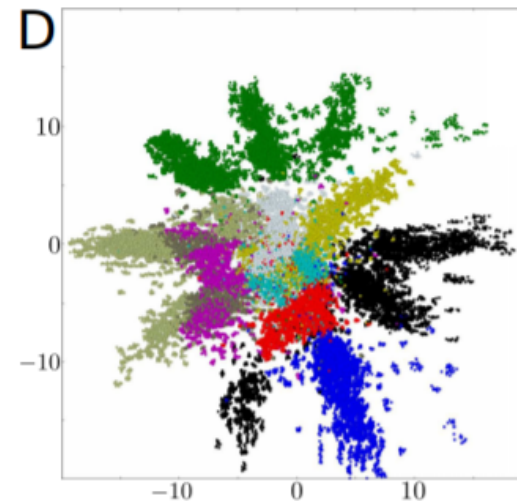
$$- \underbrace{\text{KL}(q(\mathbf{z})||p(\mathbf{z}))}_{\text{KL div. between the aggregated and prior dist.}}$$

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

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



(a) Gaussian prior



(b) GMM prior

Aggregated distributions on MNIST

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

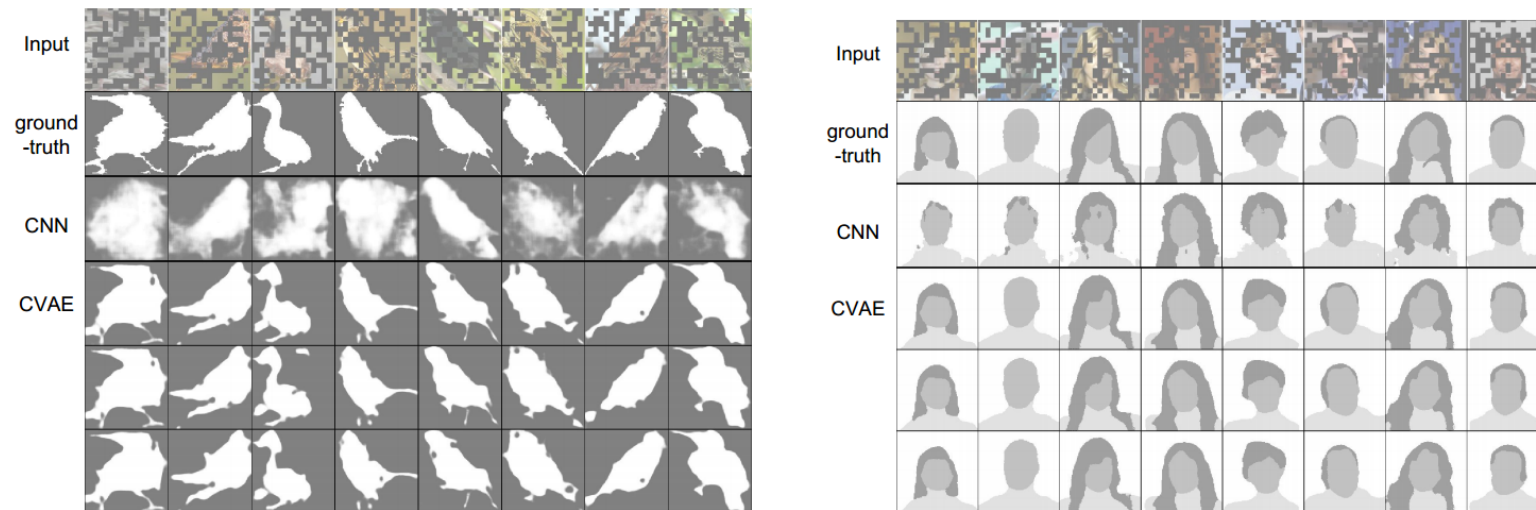
Conditional VAE (CVAE)

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

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

- Now both the encoder $q(\mathbf{Z}|\mathbf{X}, c; \boldsymbol{\theta}')$ and the decoder $p(\mathbf{X}|\mathbf{Z}, c; \boldsymbol{\theta})$ need to take c as part of their input
- How to specify the conditional prior $p(\mathbf{Z}|c)$?
 - 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 $\mathbf{Z} \sim p(\mathbf{Z}|c)$ and then passing it through the decoder $p(\mathbf{X}|\mathbf{Z}, c; \theta)$
- Learning structured outputs \mathbf{X} based on corrupted inputs c



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

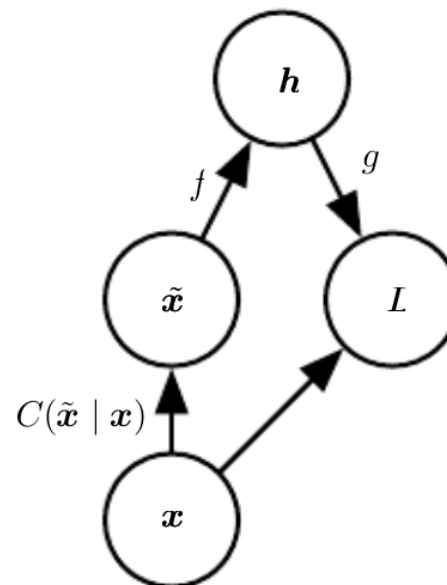
- At training time, the input image c is corrupted with part of its contents blocked randomly at different positions, and the conditional prior $p(\mathbf{Z}|c)$ is learned

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

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

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



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

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

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

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

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

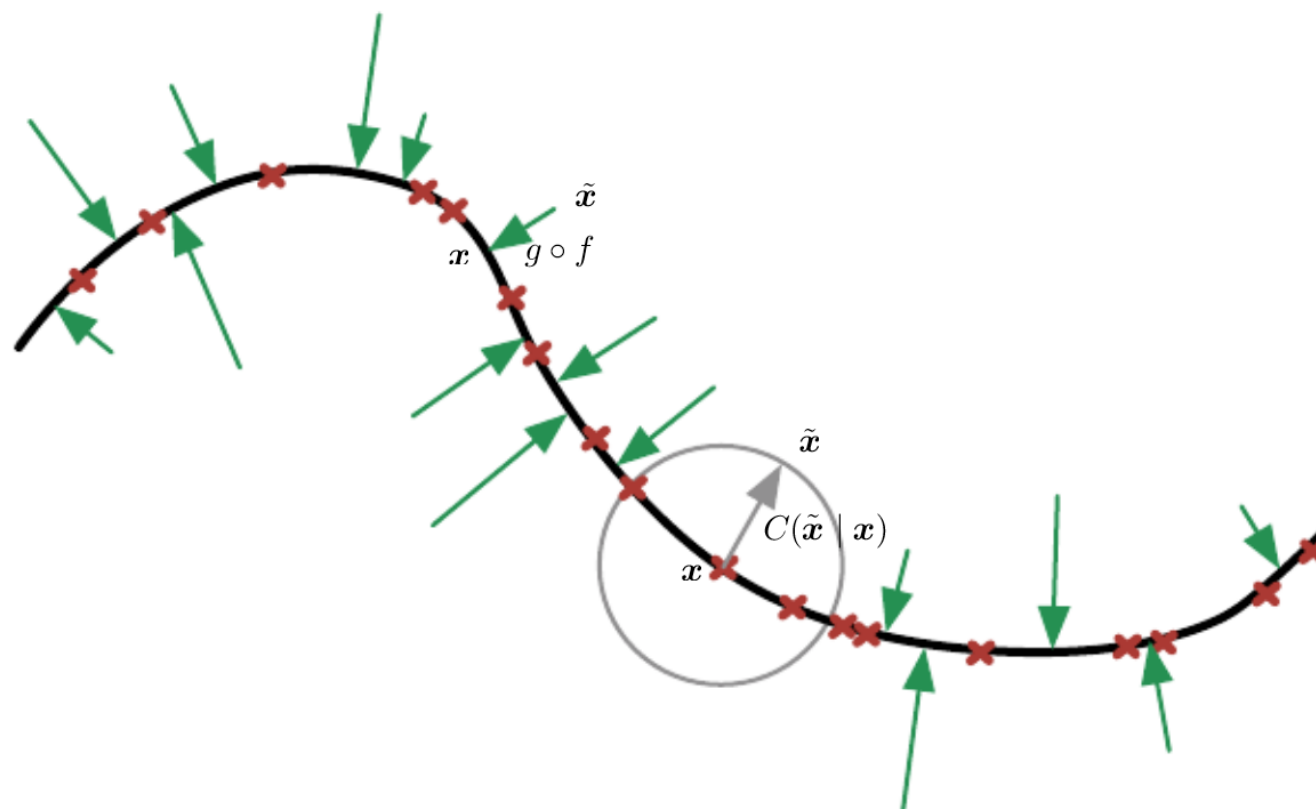
the DAE learns a vector field $(g(f(\mathbf{x})) - \mathbf{x})$ that gives estimates of **the score of the data distribution** defined as

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

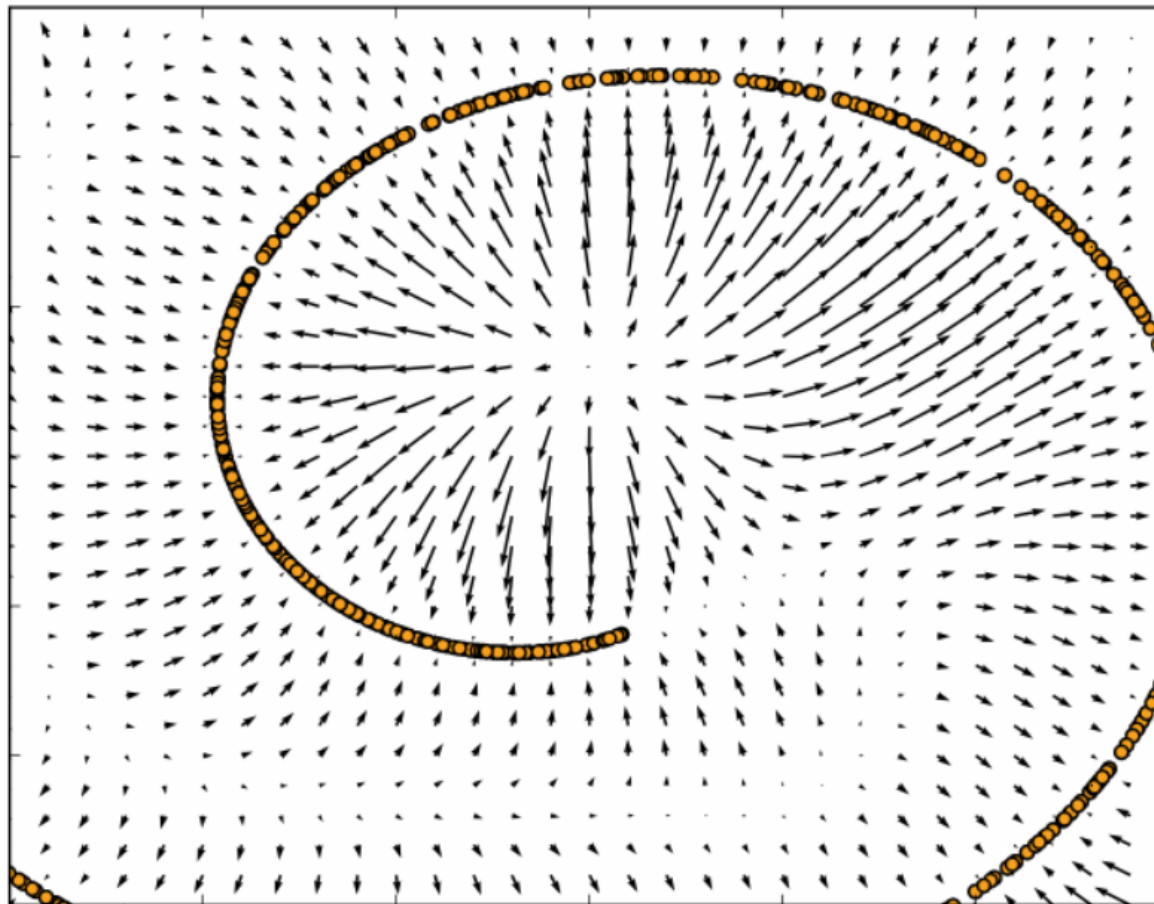
- Note that when $\|g(f(\tilde{\mathbf{x}})) - \mathbf{x}\|^2$ is minimized, we have

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

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



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



Vector field learned by a DAE
(Vector field has zeros at both maxima and minima of $p(x)$)

Sparse Autoencoders

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

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

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

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

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

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

Contractive Autoencoders (CAE)

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

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

where

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

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

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

- As such, the CAE is seen to encourage the Jacobian matrix $\partial f(\mathbf{x}_0)/\partial \mathbf{x}$ at every training point \mathbf{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

Review

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