UNIVERSITÄT BERN

Autoencoders and Variational Autoencoders

Paolo Favaro

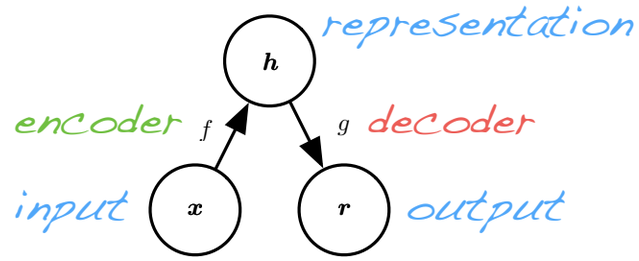
Contents

- Autoencoders
  - Denoising and stochastic AE, representation learning, manifold learning
- Variational autoencoders
- Based on **Chapter 14** of Deep Learning by Goodfellow, Bengio, Courville
- Based on “Tutorial on Variational Autoencoders” by Carl Doersch, ArXiv

Note

Autoencoders

- A network that replicates the input
- Internally it builds a **representation** of the input (e.g., as a vector)
- The network before the internal representation is the **encoder** and the network following it is the **decoder**



Note

Autoencoders

- The encoder maps input to representation

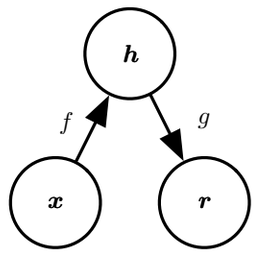
$$h = f(x)$$

- The decoder maps representation to output

$$r = g(h)$$

- Train AE with all inputs such that

$$x = g(f(x))$$



This trivially seems to just be the identity function. However, this is not so in general.

Autoencoders

- The AE is a projection operator only on the domain of the training set
- Thus, we want $x = g(f(x))$ only for $\forall x \in \Omega$
- Training is done by running SGD on

$$\min_{\theta} \sum_{x \in \Omega} L(x, g_\theta(f_\theta(x)))$$

where L is some loss function (e.g., L^2) and θ are the model parameters

The identity may not hold outside of \Omega.

Autoencoders

- In unsupervised learning the aim is to learn a model $p_{\text{model}}(x)$ that approximates $p_{\text{data}}(x)$
- Autoencoders introduce a hidden variable h and then define the model via marginalization

$$p_{\text{model}}(x) = \sum_h p_{\text{model}}(x, h)$$

$$= \sum_h p_{\text{model}}(x|h)p(h)$$

decoder

when a probability of the hidden variable is defined

In variational autoencoders the probability of h is defined as a Normal distribution. The probability of the hidden variable h can be obtained from the training set via p(h) = 1/m \sum_{i=1}^m p(h|x_i).
This requires the conditional probability p(h|x), which is provided by the encoder.

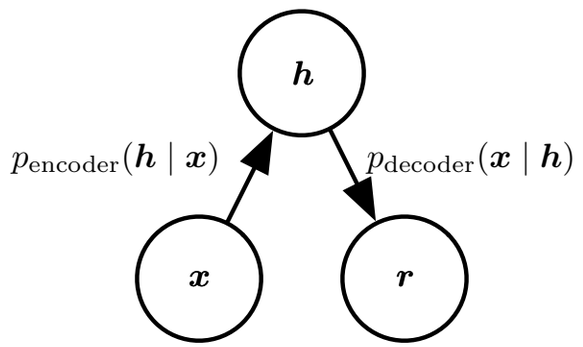
Implementation

- The encoder and the decoder can be implemented by using other standard neural networks
- For example, an encoder could use the first 5 convolutional layers of AlexNet; the decoder is often a mirroring of the encoder where convolutions with pooling or stride>1 are replaced by convolutions with fractional stride

Note

Probabilistic AE

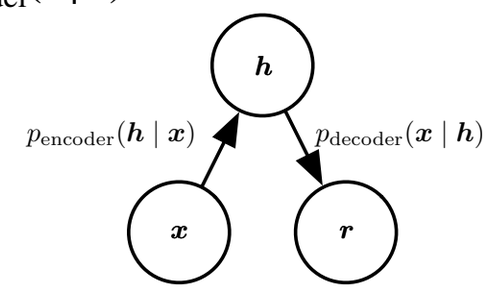
- A probabilistic model for both the encoder and the decoder corresponds to defining
  - A probability of the representation given the input
    $p_{\text{encoder}}(h|x)$
  - A probability of the reconstruction given the representation
    $p_{\text{decoder}}(x|h)$



In this case the representation h may be discrete and finite (eg categories) and we define p(h|x) (eg via softmax). Then, the output of both parts of the network are not just deterministic functions, but obtained by sampling the output probabilities.
p(x|h) could be quite high-dimensional. However, often one assumes that each element of x is conditionally independent given h.

Stochastic Encoders and Decoders

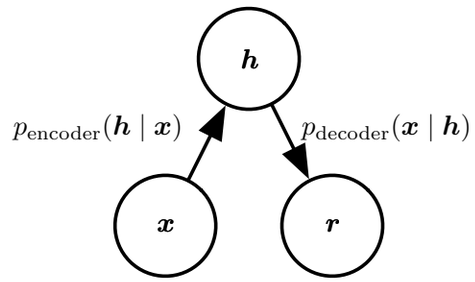
- Define a latent variable model $p_{\text{model}}(x, h)$
- Generalize the encoding function to an **encoding distribution** $p_{\text{encoder}}(h|x) = p_{\text{model}}(h|x)$
- Generalize the decoding function to a **decoding distribution** $p_{\text{decoder}}(x\,|\,h) = p_{\text{model}}(x\,|\,h)$
- Often one assumes x is conditionally independent given h



Note

Stochastic Encoders and Decoders

- If x is real-valued and has a Gaussian distribution the loss $-\log p_{\text{decoder}}(x|h)$ is L^2



- If x is
  i) binary: it is modeled with a sigmoid
  ii) discrete: it is modeled with softmax

Note

Avoiding the Trivial Identity

- **Undercomplete** autoencoders
  - h has lower dimension than x
  - f or g has low capacity (e.g., a shallow g)
  - Discard information up to h
- **Overcomplete** autoencoders
  - h has higher dimension than x
  - Must be regularized

Note

Undercomplete Autoencoders

- A way to obtain a useful representation h is to constrain it to have a smaller dimension than x
- In this case the AE is called **undercomplete**
- We force the AE to focus on the most important attributes of the training data
- A linear decoder and MSE loss L learns to map the representation to the same subspace as PCA

If the capacity of both encoder and decoder is too high, then the AE may not learn something interesting about the data. It might only learn to replicate low level and local details.

Overcomplete Autoencoders

- Choosing the representation size and the capacity of the encoder and decoder depends on the complexity of the data distribution

- A useful strategy to avoid trivial mappings is to introduce **regularization** (e.g., sparsity of the representation, smoothness of the representation, robustness to noise or missing data)

Note

Regularized Autoencoders

- Sparse autoencoders
- Denoising autoencoders
- Contractive autoencoders
- Autoencoders with dropout on the hidden layer

Note

Sparse Autoencoders

- Introduce a sparsity penalty (L^1) on the hidden representation h

$$\min_{\theta} \sum_{x \in \Omega} L(x, g_\theta(f_\theta(x))) + \Omega(f_\theta(x))$$

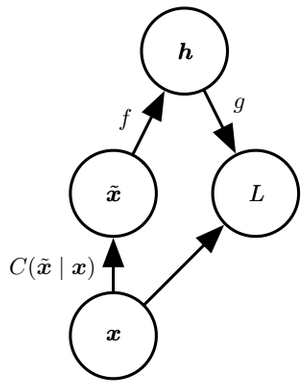
- The sparsity constraint on the representation h forces the encoder to focus on unique attributes of the dataset

- This representation is used often in classification

Note that using the representation for classification may not be a good idea. Classification categories require some invariance and focus on some specific attributes. The representation is instead general and needs to store as much information about the image to allow its reconstruction. This information may not match well to the invariance and specific attributes for classification. For example, if we just want to classify images into faces and non faces, then we do not care about details to represent houses or cars, the background, the gender of the face, glasses etc.

Denoising Autoencoders

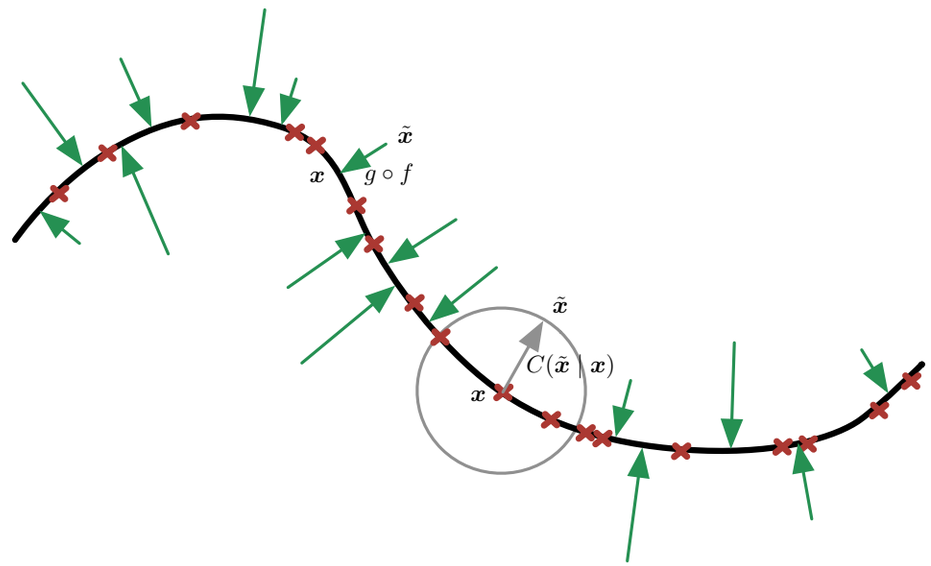
- A **denoising autoencoder** (DAE) maps noisy data to the original uncorrupted data
- Let $C(\tilde{x}|x)$ be a corruption process
- Training set $(\tilde{x}, x)$
- Optimize



$$-E_{x\sim\hat{p}_{\text{data}}} E_{\tilde{x}\sim C(\tilde{x}|x)} \log p_{\text{decoder}}(x|h = f(\tilde{x}))$$

Note

Denoising Autoencoders



Training examples are the red crosses (on a low-dimensional manifold). The corruption process generates samples \tilde x in the neighboring region (circle and grey arrow). The DAE learns a vector field mapping all noisy samples back to the manifold (shown with green arrows).

Score Matching

- In alternative to maximum likelihood, one can estimate probability distributions by matching a **score** between the data distribution and the model distribution
- Define as **score** the gradient field

$$\nabla_x \log p(x)$$

- The DAE will generate an approximation to the score by using a KDE estimate of p(x)

$$\nabla_x \log E_\epsilon[p(x-\epsilon)]$$

Score matching avoids the difficulty of dealing with the normalization factor in a distribution (the partition function). The gradient kills constants (in this case, the normalization factor is a constant). The main challenge is that we need to compute derivatives. Here we show that DAEs learn to match the above (blurred) score.
One can use the pre-trained DAE as a score (of the dataset pdf) in some loss function (for example, for score matching or as the derivative of the log prior of the data).

Score Matching

- Let $DAE(\tilde{x}) = g(f(\tilde{x}))$, then the DAE minimizes

$$E_{x \sim p_{\text{data}}} E_{\tilde{x} \sim C(\tilde{x}|x)} |DAE(\tilde{x}) - x|^2$$

which yields

$$DAE(\tilde{x}) = \tilde{x} - \frac{\sum_\epsilon \epsilon p(\tilde{x} - \epsilon) p(\epsilon)}{\sum_\epsilon p(\tilde{x} - \epsilon) p(\epsilon)}$$

by substituting $x = \tilde{x} - \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2 I_d)$

Note

Score Matching

- In this case, the DAE

$$DAE(x) = x - \frac{\sum_\epsilon \epsilon p(x-\epsilon)p(\epsilon)}{\sum_\epsilon p(x-\epsilon)p(\epsilon)}$$

yields the score

$$\frac{DAE(x) - x}{\sigma^2} = \nabla_x \log E_\epsilon[p(x-\epsilon)]$$

The difference between the DAE output and input is proportional to the approximate score.

Learning Manifolds

- Autoencoders have 2 basic requirements

  1. The internal representation must be sufficient to reconstruct the input

  2. The regularization/architectural constraints limit the response to the input

- It maps variations tangent to the manifold where data lives to variations of the hidden representation

Because of the two requirements, the autoencoder has a limited choice to what input variations it can map to output variations.

Contractive Autoencoders

- CAEs introduce a smoothness constraint on the derivatives of the encoder as regularizer

$$\Omega(h) = |\nabla_x f(x)|^2$$

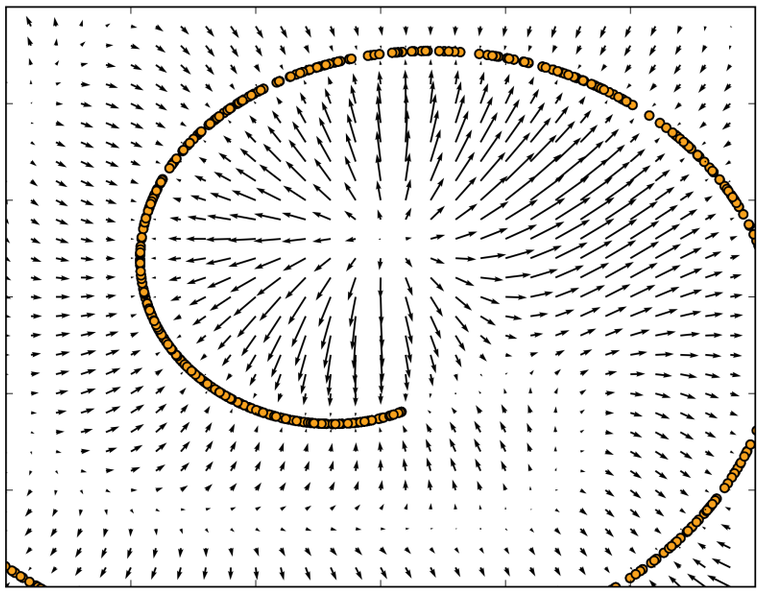
- Related to the DAE for small Gaussian input noise (the noisy reconstruction error is equivalent to the smoothness constraint)

- It maps a neighborhood of the input points to a smaller neighborhood of the output points

What is the problem with this regularizer? It can be minimized by simply scaling down the norm of the hidden code. The CAE can then easily compensate by scaling up the decoder. Thus, the system could minimize this objective without learning anything about the data. How can one avoid this behavior? One solution is to tie the weights in f with those of g (Rifai et al 2011). Another solution is to introduce a fixed normalization mapping before the output code is produced. In this case, the global scale of f cannot be changed.

Contractive Autoencoders

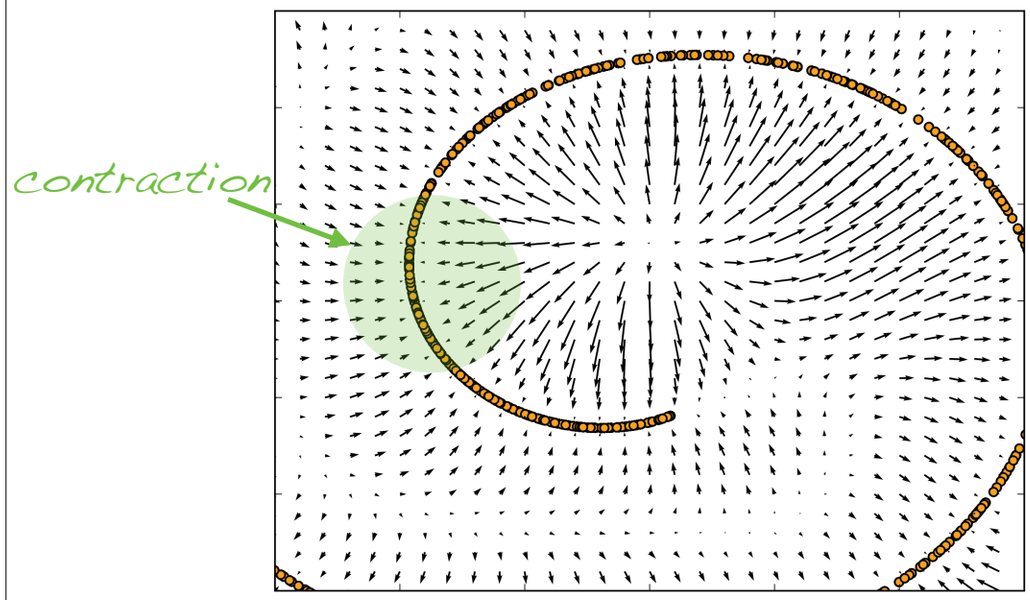
- Contraction is only local to the manifold



Note

Contractive Autoencoders

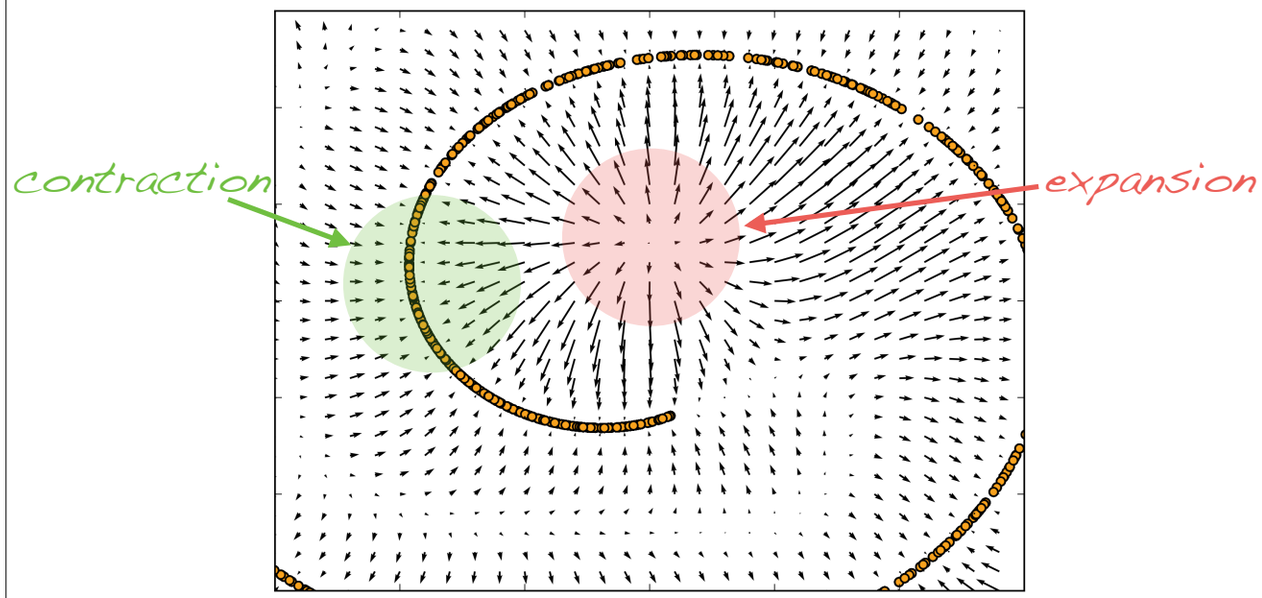
- Contraction is only local to the manifold



Note

Contractive Autoencoders

- Contraction is only local to the manifold



Note

Contractive Autoencoders

- Tangent vectors

In this comparison we can see the PCA-estimated tangent vectors from the CIFAR-10 dataset. Below we have the tangent vectors extracted by taking the SVD of the Jacobian matrix d f(x) / dx and by considering the largest singular values. One can see visually that the CAE yields more meaningful tangent vectors. They provide changes to the input that correspond to moving or changing parts of the object in the image.

Predictive Sparse Decomposition

- Impose sparsity in the code

$$\min_{h,\theta} |x - g(h;\theta)^2 + \lambda |h|_1 + \gamma |h - f(x;\theta)|^2$$

- Solve with alternating minimization
- Generates sparse codes
- Used in unsupervised feature learning for object recognition in images and video

Note

Applications

- Dimensionality reduction (representation learning)
  - More effective than PCA
  - Low-dim representations useful in other tasks (e.g., classification)
- Information retrieval (matching query to entries in a database)
  - More efficient and accurate search

A mapping to a lower dimensional space hopefully drives (semantically) related samples towards each other and keeps unrelated samples at a distance.

Generative Models

- Autoencoders can be generalized to generative models
- An important example is the **variational autoencoder**
  - Encode the moments of a Gaussian distribution

$$f(x) = \begin{bmatrix} \mu \\ \sigma \end{bmatrix}$$

  - Sample the distribution to generate data

$$\epsilon \sim \mathcal{N}(0, I_d) \qquad g(\mu + \sigma\epsilon)$$

Note

Variational Autoencoders

- **Purpose**: Unsupervised learning of complex probability distributions
- **Applications**: Generation of handwritten digits, faces, house numbers, CIFAR images, physical models of scenes, segmentation, and predicting the future from static images

- Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. ICLR, 2014.

Note

Generative Modeling

- Want to generate samples **x** of complex distributions p(**x**), such as images
- Complex because of high-dimensionality and structure (million of dimensions)
- Model needs to capture dependencies between elements (pixels)

Note

Generative Modeling

- Considerations
  - Compute an **approximation** of p(**x**):
    close to 0 when **x** is not a real image and
    close to 1 when it is.
    Not useful: if we know that **x** is unlikely, we do not know how to generate one that is likely
  - Moreover, what we want is to produce samples that are similar to those in a database, but not **identical**

If we sample p(x) then we need to keep trying until we find a high probablity. This could take a long time.
If we fit kernels around each sample in a dataset and sample them, we may end up picking the same samples over and over.

Generative Modeling

- Directly learn how to generate samples
- Samples should belong to the distribution of our training set samples
- Desiderata
  - Samples must be generated efficiently (e.g., Markov Chain Monte Carlo methods are computationally inefficient)
  - No strong assumptions on the data distribution
  - No strong approximations of the generative model

Note

Latent Variables

- A probability model can be described via the interaction of several random variables

- The variables that we do not directly observe are called **latent variables**

- **Example**
  - We want to generate images of the digits 0-9
  - Define as latent variable **z** the digit
  - The observed variable **x** is an image corresponding to the digit **z**.

Note

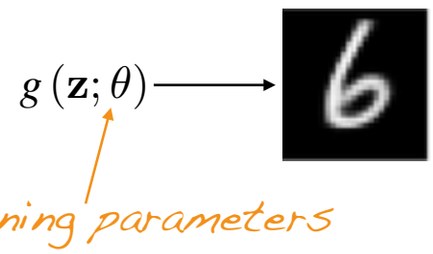
Latent Variables

- **Example**: Digit image generation

  Sample $\mathbf{z}$ from p($\mathbf{z}$)

  $\mathbf{z}$={6, stroke thickness, stroke angle, …}

  Generate image with the generative model



Note

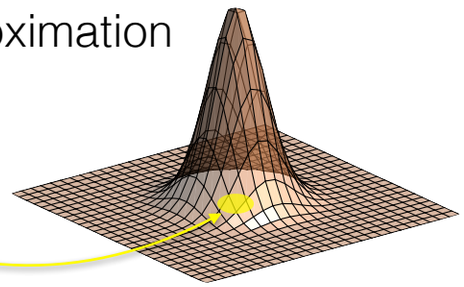
Maximum Likelihood Model

- We aim to approximate the distribution of $\mathbf{x}$ via

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

where we use the following approximation

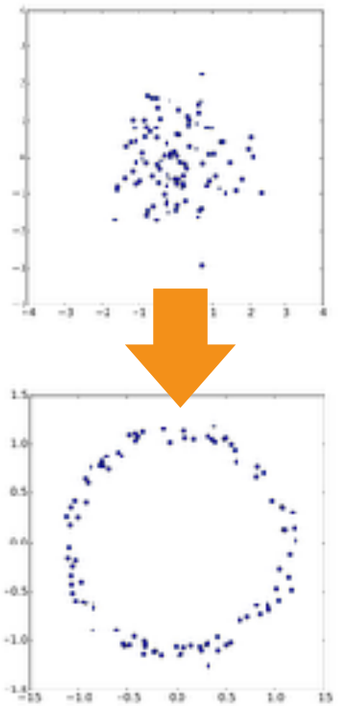
$$p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x};\, g(\mathbf{z};\theta), \sigma^2 I_d)$$



The model f provides the mean image but around it the samples should also be plausible. Notice that this does not mean that p(x) is Gaussian. The integral in z needs to deal with the nonlinear mapping of the mean of p(x|z) due to g.
We aim to maximize the likelihood p(x) with respect to the parameters \theta on a given dataset of data samples x. Notice that if we have \sigma = 0, there would be no iterative method that can converge, since the p(x_i|z) would always be zero (it would be very unlikely that the initial f model can generate samples close to the given data).

Latent Variables Model

- What can we choose for p($\mathbf{z}$)?
  The **Normal distribution**
- No restrictions: There always exists a transformation of a Gaussian in d dimensions to any other d-dimensional distribution
- The model f will apply the needed transformation (which will be learned)



Note

Naïve Maximum Likelihood

- A simple approach to obtain the model is then to solve the maximum likelihood problem on a dataset

$$\max_{\theta} p(\mathbf{x}; \theta)$$

where

$$p(\mathbf{x}; \theta) = \frac{1}{m} \sum_{i} p\left(\mathbf{x} | \mathbf{z}^{(i)}; \theta\right)$$

but because of the high dimensionality of $\mathbf{x}$ this requires a lot of samples to give a reasonable approximation

A meaningful similarity of two digits in grayscale space requires a very tight \sigma. Then, we need lots of samples in z to hope to obtain a meaningful x. To avoid relying on luck, we can instead build a function that gives us a good guess for a z that would yield a given x!

Learning What Samples to Sample

- To avoid sampling $\mathbf{z}$ where there is no need, i.e., where p($\mathbf{x}$|$\mathbf{z}$)=0, we learn a function that gives us the distribution of $\mathbf{z}$ that yields $\mathbf{x}$!

- Compute Q($\mathbf{z}$) as the approximation of p($\mathbf{z}$|$\mathbf{x}$) via the minimization of the Kullback-Leibler distance

$$D\left(Q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x})\right)$$

  which is equivalent to

$$E_{\mathbf{z}\sim Q}\left[\log Q(\mathbf{z}) - \log p(\mathbf{x}|\mathbf{z}) - \log p(\mathbf{z})\right] + \log p(\mathbf{x})$$

Note that x is given (our dataset).

Learning What Samples to Sample

- Since the equation must hold for each $\mathbf{x}$, we can change Q at each $\mathbf{x}$ to get a better approximation

$$\log p(\mathbf{x}) - D(Q(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}|\mathbf{x})) = E_{\mathbf{z}\sim Q}[\log p(\mathbf{x}|\mathbf{z})] - D(Q(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))$$

encoder (data to latent)

decoder (latent to data)

Note

Learning What Samples to Sample

- With maximum likelihood we maximize

$$\log p(\mathbf{x}) - D(Q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))$$

so that

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

is maximized and

$$D(Q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))$$

is minimized, yielding Q($\mathbf{z}|\mathbf{x}$) = p($\mathbf{z}|\mathbf{x}$)

Note

Learning What Samples to Sample

- Maximizing

$$\log p(\mathbf{x}) - D(Q(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}|\mathbf{x}))$$

  is equivalent to maximizing

$$E_{\mathbf{z}\sim Q}[\log p(\mathbf{x}|\mathbf{z})] - D(Q(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))$$

- The next step is to choose the explicit form of Q($\mathbf{z}|\mathbf{x}$)

- We choose a Gaussian

$$Q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \mu(\mathbf{x};\theta), \Sigma(\mathbf{x};\theta))$$

By working on the right hand side of the equation we have a simpler optimization problem (all terms are numerically tractable). The Gaussian model for the distribution Q uses mean and covariance obtained from a neural network.

Learning What Samples to Sample

- Because of the Gaussian assumption on Q, we can compute the Kullback-Leibler distance

$$D(Q(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))$$

in closed form

$$\frac{1}{2}\left(\mathrm{tr}\Sigma(\mathbf{x}) + \mu(\mathbf{x})^\top\mu(\mathbf{x}) - k - \log\det\Sigma(\mathbf{x})\right)$$

dimension of $\mathbf{z}$

By working on the right hand side of the equation we have a simpler optimization problem (all terms are numerically tractable). The Gaussian model for the distribution Q uses mean and covariance obtained from a neural network.

The Reparametrization Trick

- We need to approximate the term

$$E_{\mathbf{z}\sim Q}[\log p(\mathbf{x}|\mathbf{z})]$$

- How to sample $Q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \mu(\mathbf{x};\theta), \Sigma(\mathbf{x};\theta))$?
- We can first sample $\epsilon \sim \mathcal{N}(0, I_d)$ and then let

$$\mathbf{z} = \mu(\mathbf{x}) + \Sigma(\mathbf{x})^{1/2}\epsilon$$

The problem is to compute derivatives with respect to the model parameters for backpropagation.

The Reparametrization Trick

- Then, the term

$$E_{\mathbf{z}\sim Q}[\log p(\mathbf{x}|\mathbf{z})]$$

becomes

$$E_{\epsilon\sim\mathcal{N}}\left[\log p(\mathbf{x}|\mathbf{z}=\mu(\mathbf{x};\theta)+\Sigma(\mathbf{x};\theta)^{1/2}\epsilon)\right]$$

- If we use a single sample approximation we obtain the explicit form (by ignoring the constants)

$$-\frac{\left\|\mathbf{x}-g\left(\mu(\mathbf{x};\theta)+\Sigma(\mathbf{x};\theta)^{1/2}\epsilon;\theta\right)\right\|^2}{2\sigma^2}$$

Note

VAE Loss Function

- By putting all together we obtain the following loss function (after simplifying)

$$\min_{\theta} \sum_{i} \Bigg[\left\| \mathbf{x}^{(i)} - g\left(\mu(\mathbf{x}^{(i)};\theta) + \Sigma(\mathbf{x}^{(i)};\theta)^{1/2}\epsilon;\theta\right) \right\|^2 + \sigma^2 \mathrm{tr}\ \Sigma(\mathbf{x}^{(i)};\theta) + \sigma^2 \mu(\mathbf{x}^{(i)};\theta)^\top \mu(\mathbf{x}^{(i)};\theta) - \sigma^2 \log\det \Sigma(\mathbf{x}^{(i)};\theta) \Bigg]$$

The first term is similar to an autoencoder where we add noise to the latent representation (the mean vector). The other terms are regularization parameters. They favor means with a small L2 norm and small covariances.

Using the Model

- After training we obtain the parameters θ and we use the decoder after sampling $\mathbf{z}$ from the Normal distribution

$$\mathbf{z} \sim \mathcal{N}(0, I_d) \longrightarrow g(\mathbf{z}; \theta) \longrightarrow$$

Note

VAE on MNIST

Note

Limitations

- VAEs produce samples that typically have a lower quality than those of other generative models
- A possible limitation is the Gaussian assumption for $p(\mathbf{x}|\mathbf{z})$
- Indeed it is not true that real images can be obtained by adding Gaussian noise to a real image
- This might be the reason why generated samples tend to be blurry