Week 1 Overview: Variational Autoencoders

Shikhar Agrawal and Mayank Jain
Supervised by: Prof. Suyash P. Awate

Indian Institute of Technology Bombay

08 August, 2022

Table of Contents

Table of Contents

Introduction to Autoencoders

- An Autoencoder is a type of artificial neural network used to learn efficient codings of unlabeled data (unsupervised learning). The encoding is validated and refined by attempting to regenerate the input from the encoding

Introduction to Autoencoders

- An Autoencoder is a type of artificial neural network used to learn efficient codings of unlabeled data (unsupervised learning). The encoding is validated and refined by attempting to regenerate the input from the encoding
- Here the network is trained to ignore insignificant data("noise") and to reduce the dimensionality of input data

Introduction to Autoencoders

- An Autoencoder is a type of artificial neural network used to learn efficient codings of unlabeled data (unsupervised learning). The encoding is validated and refined by attempting to regenerate the input from the encoding
- Here the network is trained to ignore insignificant data(”noise”) and to reduce the dimensionality of input data
- Variants exist, aiming to force the learned representations to assume useful properties. Examples are regularized autoencoders (Sparse, Denoising and Contractive), which are effective in learning representations for subsequent classification tasks, and Variational autoencoders, with applications as generative models

Introduction to Autoencoders

- An Autoencoder is a type of artificial neural network used to learn efficient codings of unlabeled data (unsupervised learning). The encoding is validated and refined by attempting to regenerate the input from the encoding
- Here the network is trained to ignore insignificant data("noise") and to reduce the dimensionality of input data
- Variants exist, aiming to force the learned representations to assume useful properties. Examples are regularized autoencoders (Sparse, Denoising and Contractive), which are effective in learning representations for subsequent classification tasks, and Variational autoencoders, with applications as generative models
- Autoencoders are applied to many problems, including facial recognition, feature detection, anomaly detection and acquiring the meaning of words. Autoencoders are also generative models which can randomly generate new data that is similar to the input data (training data)

Mathematical Principles

Formal Definition

An autoencoder is defined by the following components. Two sets: the space of decoded messages $\mathcal{X}$; the space of encoded messages $\mathcal{Z}$. Almost always, both $\mathcal{X}$ and $\mathcal{Z}$ are Euclidean spaces, that is, $\mathcal{X} = \mathbb{R}^m$, $\mathcal{Z} = \mathbb{R}^n$ for some m, n. Two parametrized families of functions: the encoder family $\mathcal{E}_\phi : \mathcal{X} \longrightarrow \mathcal{Z}$, parametrized by ϕ ; the decoder family $\mathcal{D}_\theta : \mathcal{Z} \longrightarrow \mathcal{X}$, parametrized by θ

Mathematical Principles

Formal Definition

An autoencoder is defined by the following components. Two sets: the space of decoded messages $\mathcal{X}$; the space of encoded messages $\mathcal{Z}$. Almost always, both $\mathcal{X}$ and $\mathcal{Z}$ are Euclidean spaces, that is, $\mathcal{X} = \mathbb{R}^m$, $\mathcal{Z} = \mathbb{R}^n$ for some m, n. Two parametrized families of functions: the encoder family $\mathcal{E}_\phi : \mathcal{X} \longrightarrow \mathcal{Z}$, parametrized by ϕ ; the decoder family $\mathcal{D}_\theta : \mathcal{Z} \longrightarrow \mathcal{X}$, parametrized by θ

- For any $x \in \mathcal{X}$, we usually write $z = \mathcal{E}_\phi(x)$, and refer to it as the code, the latent variable, latent representation, latent vector, etc. Conversely, for any $z \in \mathcal{Z}$, we usually write $x' = \mathcal{D}_\theta(z)$, and refer to it as the (decoded) message

Mathematical Principles

Formal Definition

An autoencoder is defined by the following components. Two sets: the space of decoded messages $\mathcal{X}$; the space of encoded messages $\mathcal{Z}$. Almost always, both $\mathcal{X}$ and $\mathcal{Z}$ are Euclidean spaces, that is, $\mathcal{X} = \mathbb{R}^m$, $\mathcal{Z} = \mathbb{R}^n$ for some m, n. Two parametrized families of functions: the encoder family $\mathcal{E}_\phi : \mathcal{X} \longrightarrow \mathcal{Z}$, parametrized by ϕ ; the decoder family $\mathcal{D}_\theta : \mathcal{Z} \longrightarrow \mathcal{X}$, parametrized by θ

- For any $x \in \mathcal{X}$, we usually write $z = \mathcal{E}_\phi(x)$, and refer to it as the code, the latent variable, latent representation, latent vector, etc. Conversely, for any $z \in \mathcal{Z}$, we usually write $x' = \mathcal{D}_\theta(z)$, and refer to it as the (decoded) message
- Usually, both the encoder and the decoder are defined as multilayer perceptrons

Training an Autoencoder

- An autoencoder, by itself, is simply a tuple of two functions. To judge its quality, we need a task. A task is defined by a reference probability distribution μ_{ref} over $\mathcal{X}$, and a "reconstruction quality" function $d : \mathcal{X} \times \mathcal{X} \longrightarrow [0, \infty]$, such that $d(x, x')$ measures how much x' differs from x.

Training an Autoencoder

- An autoencoder, by itself, is simply a tuple of two functions. To judge its quality, we need a task. A task is defined by a reference probability distribution μ_{ref} over $\mathcal{X}$, and a "reconstruction quality" function $d : \mathcal{X} \times \mathcal{X} \longrightarrow [0, \infty]$, such that $d(x, x')$ measures how much x' differs from x.
- With those, we can define the loss function for the autoencoder as $L(\theta, \phi) = \mathbb{E}_{x \sim \mu_{ref}}[d(x, \mathcal{D}_\theta(\mathcal{E}_\phi(x)))]$

Training an Autoencoder

- An autoencoder, by itself, is simply a tuple of two functions. To judge its quality, we need a task. A task is defined by a reference probability distribution μ_{ref} over $\mathcal{X}$, and a "reconstruction quality" function $d : \mathcal{X} \times \mathcal{X} \longrightarrow [0, \infty]$, such that $d(x, x')$ measures how much x' differs from x.
- With those, we can define the loss function for the autoencoder as $L(\theta, \phi) = \mathbb{E}_{x \sim \mu_{ref}}[d(x, \mathcal{D}_\theta(\mathcal{E}_\phi(x)))]$
- The optimal autoencoder for the given task (μ_{ref}, d) is then $\arg\min_{\theta,\phi} L(\theta, \phi)$. The search for the optimal autoencoder can be accomplished by any mathematical optimization technique, but usually by gradient descent.

Training an Autoencoder

- An autoencoder, by itself, is simply a tuple of two functions. To judge its quality, we need a task. A task is defined by a reference probability distribution μ_{ref} over $\mathcal{X}$, and a "reconstruction quality" function $d : \mathcal{X} \times \mathcal{X} \longrightarrow [0, \infty]$, such that $d(x, x')$ measures how much x' differs from x.
- With those, we can define the loss function for the autoencoder as $L(\theta, \phi) = \mathbb{E}_{x \sim \mu_{ref}}[d(x, \mathcal{D}_\theta(\mathcal{E}_\phi(x)))]$
- The optimal autoencoder for the given task (μ_{ref}, d) is then $\arg\min_{\theta,\phi} L(\theta, \phi)$. The search for the optimal autoencoder can be accomplished by any mathematical optimization technique, but usually by gradient descent.
- In most situations, the reference distribution is just the empirical distribution given by a dataset $\{x_1, ..., x_N\} \subset \mathcal{X}$, so that $\mu_{ref} = \frac{1}{N} \sum_{i=1}^{N} \delta_{x_i}$ and the quality function is just L_2 loss: $d(x, x') = ||x - x'||_2^2$

Training an Autoencoder

- An autoencoder, by itself, is simply a tuple of two functions. To judge its quality, we need a task. A task is defined by a reference probability distribution μ_{ref} over $\mathcal{X}$, and a "reconstruction quality" function $d : \mathcal{X} \times \mathcal{X} \longrightarrow [0, \infty]$, such that $d(x, x')$ measures how much x' differs from x.
- With those, we can define the loss function for the autoencoder as $L(\theta, \phi) = \mathbb{E}_{x \sim \mu_{ref}}[d(x, \mathcal{D}_\theta(\mathcal{E}_\phi(x)))]$
- The optimal autoencoder for the given task (μ_{ref}, d) is then $\arg\min_{\theta,\phi} L(\theta, \phi)$. The search for the optimal autoencoder can be accomplished by any mathematical optimization technique, but usually by gradient descent.
- In most situations, the reference distribution is just the empirical distribution given by a dataset $\{x_1, ..., x_N\} \subset \mathcal{X}$, so that $\mu_{ref} = \frac{1}{N}\sum_{i=1}^{N} \delta_{x_i}$ and the quality function is just L_2 loss: $d(x, x') = ||x - x'||_2^2$
- Then the problem of searching for the optimal autoencoder is just a least-squares optimization: $\min_{\theta,\phi} L(\theta, \phi)$, where $L(\theta, \phi) = \frac{1}{N}\sum_{i=1}^{N} ||x_i - \mathcal{D}_\theta(\mathcal{E}_\phi(x_i))||_2^2$

Interpretation of an Autoencoder

- An optimal autoencoder would perform as close to perfect reconstruction as possible, with 'close to perfect' defined by the reconstruction quality function d

Interpretation of an Autoencoder

- An optimal autoencoder would perform as close to perfect reconstruction as possible, with 'close to perfect' defined by the reconstruction quality function d
- The simplest way to perform the copying task perfectly would be to duplicate the signal. To suppress this behavior, the code space $\mathcal{Z}$ usually has fewer dimensions than the message space $\mathcal{X}$. Such an autoencoder is undercomplete.

Interpretation of an Autoencoder

- An optimal autoencoder would perform as close to perfect reconstruction as possible, with 'close to perfect' defined by the reconstruction quality function d
- The simplest way to perform the copying task perfectly would be to duplicate the signal. To suppress this behavior, the code space $\mathcal{Z}$ usually has fewer dimensions than the message space $\mathcal{X}$. Such an autoencoder is undercomplete.
- At the limit of an ideal undercomplete autoencoder, every possible code z in the code space is used to encode a message x that really appears in the distribution μ_{ref}, and the decoder is also perfect: $\mathcal{D}_\theta(\mathcal{E}_\phi(x)) = x$. This ideal autoencoder can then be used to generate messages indistinguishable from real messages, by feeding its decoder arbitrary code z and obtaining $\mathcal{D}_\theta(z)$, which is a message that really appears in the distribution μ_{ref}

Table of Contents

Sparse Autoencoder (SAE)

- Here the codes $\mathcal{D}_\phi(x)$ for messages tend to be sparse codes, that is, $\mathcal{D}_\phi(x)$ is close to 0 in most entries.

Sparse Autoencoder (SAE)

- Here the codes $\mathcal{D}_\phi(x)$ for messages tend to be sparse codes, that is, $\mathcal{D}_\phi(x)$ is close to 0 in most entries.
- There are two main ways to enforce sparsity. One way is to simply clamp all but the highest-k activations of the latent code to zero. This is the k-sparse autoencoder.

Sparse Autoencoder (SAE)

- Here the codes $\mathcal{D}_\phi(x)$ for messages tend to be sparse codes, that is, $\mathcal{D}_\phi(x)$ is close to 0 in most entries.
- There are two main ways to enforce sparsity. One way is to simply clamp all but the highest-k activations of the latent code to zero. This is the k-sparse autoencoder.
- The k-sparse autoencoder inserts the following "k-sparse function" in the latent layer of a standard autoencoder: $f_k(x_1, ..., x_N) = (x_1 b_1, ..., x_n b_n)$, where $b_i = 1$ if $|x_i|$ ranks in the top k, and 0 otherwise

Sparse Autoencoder (SAE)

- Here the codes $\mathcal{D}_\phi(x)$ for messages tend to be sparse codes, that is, $\mathcal{D}_\phi(x)$ is close to 0 in most entries.
- There are two main ways to enforce sparsity. One way is to simply clamp all but the highest-k activations of the latent code to zero. This is the k-sparse autoencoder.
- The k-sparse autoencoder inserts the following "k-sparse function" in the latent layer of a standard autoencoder: $f_k(x_1, ..., x_N) = (x_1 b_1, ..., x_n b_n)$, where $b_i = 1$ if $|x_i|$ ranks in the top k, and 0 otherwise
- Backpropagating through f_k is simple: set gradient to 0 for $b_i = 0$ entries, and keep gradient for $b_i = 1$ entries. This is essentially a generalized ReLU function.

Relaxed k-sparse Autoencoder

- Instead of forcing sparsity, we add a sparsity regularization loss, then optimize for,

  $\min_{\theta,\phi} L(\theta,\phi) + \lambda L_{sparsity}(\theta,\phi)$ where $\lambda > 0$ measures how much sparsity we want to enforce.

Relaxed k-sparse Autoencoder

- Instead of forcing sparsity, we add a sparsity regularization loss, then optimize for,

  $\min_{\theta,\phi} L(\theta, \phi) + \lambda L_{sparsity}(\theta, \phi)$ where $\lambda > 0$ measures how much sparsity we want to enforce.
- Let the autoencoder architecture have $\mathcal{K}$ layers. To define a sparsity regularization loss, we need a "desired" sparsity $\hat{\rho}_k$ for each layer, a weight w_k for how much to enforce each sparsity, and a function $s : [0,1] \times [0,1] \longrightarrow [0,\infty]$ to measure much two sparsities differ.

Relaxed k-sparse Autoencoder

- Instead of forcing sparsity, we add a sparsity regularization loss, then optimize for,

  $\min_{\theta,\phi} L(\theta, \phi) + \lambda L_{sparsity}(\theta, \phi)$ where $\lambda > 0$ measures how much sparsity we want to enforce.
- Let the autoencoder architecture have $\mathcal{K}$ layers. To define a sparsity regularization loss, we need a "desired" sparsity $\hat{\rho}_k$ for each layer, a weight w_k for how much to enforce each sparsity, and a function $s : [0, 1] \times [0, 1] \longrightarrow [0, \infty]$ to measure much two sparsities differ.
- For each input x, let the actual sparsity of activation in each layer k be $\rho_k(x) = \frac{1}{n} \sum_{i=1}^{n} a_{k,i}(x)$, where $a_{k,i}(x)$ is the activation in the i^{th} neuron of the k^{th} layer upon input x

Relaxed k-sparse Autoencoder

- Instead of forcing sparsity, we add a sparsity regularization loss, then optimize for,

  $\min_{\theta,\phi} L(\theta, \phi) + \lambda L_{sparsity}(\theta, \phi)$ where $\lambda > 0$ measures how much sparsity we want to enforce.
- Let the autoencoder architecture have $\mathcal{K}$ layers. To define a sparsity regularization loss, we need a "desired" sparsity $\hat{\rho}_k$ for each layer, a weight w_k for how much to enforce each sparsity, and a function $s : [0,1] \times [0,1] \longrightarrow [0,\infty]$ to measure much two sparsities differ.
- For each input x, let the actual sparsity of activation in each layer k be $\rho_k(x) = \frac{1}{n} \sum_{i=1}^{n} a_{k,i}(x)$, where $a_{k,i}(x)$ is the activation in the i^{th} neuron of the k^{th} layer upon input x
- The sparsity regularization loss for the entire autoencoder is the expected weighted sum of sparsity losses: $L_{sparsity}(\theta, \phi) = \mathbb{E}_{x \sim \mu_{\mathcal{X}}} \left[\sum_{k \in 1:K} w_k s(\hat{\rho}_k, \rho_k(x)) \right]$.

Relaxed k-sparse Autoencoder

- Instead of forcing sparsity, we add a sparsity regularization loss, then optimize for,

  $\min_{\theta,\phi} L(\theta, \phi) + \lambda L_{sparsity}(\theta, \phi)$ where $\lambda > 0$ measures how much sparsity we want to enforce.
- Let the autoencoder architecture have $\mathcal{K}$ layers. To define a sparsity regularization loss, we need a "desired" sparsity $\hat{\rho_k}$ for each layer, a weight w_k for how much to enforce each sparsity, and a function $s : [0, 1] \times [0, 1] \longrightarrow [0, \infty]$ to measure much two sparsities differ.
- For each input x, let the actual sparsity of activation in each layer k be $\rho_k(x) = \frac{1}{n} \sum_{i=1}^{n} a_{k,i}(x)$, where $a_{k,i}(x)$ is the activation in the i^{th} neuron of the k^{th} layer upon input x
- The sparsity regularization loss for the entire autoencoder is the expected weighted sum of sparsity losses: $L_{sparsity}(\theta, \phi) = \mathbb{E}_{x \sim \mu_{\mathcal{X}}} \left[\sum_{k \in 1:K} w_k s(\hat{\rho_k}, \rho_k(x)) \right]$.
- Typically, the function s is either the Kullback-Leibler (KL) divergence, as $s(\rho, \hat{\rho}) = KL(\rho || \hat{\rho}) = \rho log \frac{\rho}{\hat{\rho}} + (1 - \rho) log \frac{1-\rho}{1-\hat{\rho}}$ or the L_2 as $s(\rho, \hat{\rho}) = |\rho - \hat{\rho}|^2$ or the L_1 loss as $s(\rho, \hat{\rho}) = |\rho - \hat{\rho}|$

Relaxed k-sparse Autoencoder

- Instead of forcing sparsity, we add a sparsity regularization loss, then optimize for,
  $\min_{\theta,\phi} L(\theta,\phi) + \lambda L_{sparsity}(\theta,\phi)$ where $\lambda > 0$ measures how much sparsity we want to enforce.
- Let the autoencoder architecture have $\mathcal{K}$ layers. To define a sparsity regularization loss, we need a "desired" sparsity $\hat{\rho}_k$ for each layer, a weight w_k for how much to enforce each sparsity, and a function $s : [0,1] \times [0,1] \longrightarrow [0,\infty]$ to measure much two sparsities differ.
- For each input x, let the actual sparsity of activation in each layer k be $\rho_k(x) = \frac{1}{n}\sum_{i=1}^{n} a_{k,i}(x)$, where $a_{k,i}(x)$ is the activation in the i^{th} neuron of the k^{th} layer upon input x
- The sparsity regularization loss for the entire autoencoder is the expected weighted sum of sparsity losses: $L_{sparsity}(\theta,\phi) = \mathbb{E}_{x\sim\mu_{\mathcal{X}}}\left[\sum_{k\in 1:K} w_k s(\hat{\rho}_k, \rho_k(x))\right]$.
- Typically, the function s is either the Kullback-Leibler (KL) divergence, as $s(\rho,\hat{\rho}) = KL(\rho||\hat{\rho}) = \rho log\frac{\rho}{\hat{\rho}} + (1-\rho)log\frac{1-\rho}{1-\hat{\rho}}$ or the L_2 as $s(\rho,\hat{\rho}) = |\rho-\hat{\rho}|^2$ or the L_1 loss as $s(\rho,\hat{\rho}) = |\rho-\hat{\rho}|$
- One can define the sparsity regularization loss as $L_{sparsity}(\theta,\phi) = \mathbb{E}_{x\sim\mu_{\mathcal{X}}}\left[\sum_{k\in 1:K} w_k||h_k||\right]$, where h_k is the activation vector in the k^{th} layer of the autoencoder.

Denoising Autoencoder (DAE)

- Denoising autoencoders (DAE) try to achieve a good representation by changing the reconstruction criterion. A DAE is defined by adding a noise process to the standard autoencoder

Denoising Autoencoder (DAE)

- Denoising autoencoders (DAE) try to achieve a good representation by changing the reconstruction criterion. A DAE is defined by adding a noise process to the standard autoencoder
- A noise process is defined by a probability distribution $\mu_{\mathcal{T}}$ over functions $\mathcal{T} : \mathcal{X} \longrightarrow \mathcal{X}$. That is, the function $\mathcal{X}$ takes a message $x \in \mathcal{X}$, and corrupts it to a noisy version $\mathcal{T}(x)$. The function $\mathcal{T}$ is selected randomly, with a probability distribution $\mu_{\mathcal{T}}$

Denoising Autoencoder (DAE)

- Denoising autoencoders (DAE) try to achieve a good representation by changing the reconstruction criterion. A DAE is defined by adding a noise process to the standard autoencoder
- A noise process is defined by a probability distribution $\mu_{\mathcal{T}}$ over functions $\mathcal{T} : \mathcal{X} \longrightarrow \mathcal{X}$. That is, the function $\mathcal{X}$ takes a message $x \in \mathcal{X}$, and corrupts it to a noisy version $\mathcal{T}(x)$. The function $\mathcal{T}$ is selected randomly, with a probability distribution $\mu_{\mathcal{T}}$
- Given a task (μ_{ref}, d), the problem of training a DAE is the optimization problem: $\min_{\theta,\phi} L(\theta, \phi) = \mathbb{E}_{x \sim \mu_{\mathcal{X}}, \mathcal{T} \sim \mu_{\mathcal{T}}} [d(x, (\mathcal{D}_\theta \circ \mathcal{E}_\phi \circ \mathcal{T})(x))]$

Denoising Autoencoder (DAE)

- Denoising autoencoders (DAE) try to achieve a good representation by changing the reconstruction criterion. A DAE is defined by adding a noise process to the standard autoencoder
- A noise process is defined by a probability distribution $\mu_{\mathcal{T}}$ over functions $\mathcal{T} : \mathcal{X} \longrightarrow \mathcal{X}$. That is, the function $\mathcal{X}$ takes a message $x \in \mathcal{X}$, and corrupts it to a noisy version $\mathcal{T}(x)$. The function $\mathcal{T}$ is selected randomly, with a probability distribution $\mu_{\mathcal{T}}$
- Given a task (μ_{ref}, d), the problem of training a DAE is the optimization problem: $\min\limits_{\theta,\phi} L(\theta, \phi) = \mathbb{E}_{x \sim \mu_{\mathcal{X}}, \mathcal{T} \sim \mu_{\mathcal{T}}} [d(x, (\mathcal{D}_\theta \circ \mathcal{E}_\phi \circ \mathcal{T})(x))]$
- The use of DAE depends on two assumptions: There exist representations to the messages that are relatively stable and robust to the type of noise we are likely to encounter; and the said representations capture structures in the input distribution that are useful for our purposes.

Contractive Autoencoders (CAE)

- A contractive autoencoder adds the contractive regularization loss to the standard autoencoder loss: $\min_{\theta,\phi} L(\theta,\phi) + \lambda L_{contractive}(\theta,\phi)$, where $\lambda > 0$ measures how much contractive-ness we want to enforce

Contractive Autoencoders (CAE)

- A contractive autoencoder adds the contractive regularization loss to the standard autoencoder loss: $\min_{\theta,\phi} L(\theta, \phi) + \lambda L_{contractive}(\theta, \phi)$, where $\lambda > 0$ measures how much contractive-ness we want to enforce
- The contractive regularization loss itself is defined as the expected Frobenius norm of the Jacobian matrix of the encoder activations with respect to the input: $L_{contractive}(\theta, \phi) = \mathbb{E}_{x \sim \mu_{ref}} ||\nabla_x \mathcal{E}_\phi(x)||_F^2$

Contractive Autoencoders (CAE)

- A contractive autoencoder adds the contractive regularization loss to the standard autoencoder loss: $\min_{\theta,\phi} L(\theta, \phi) + \lambda L_{contractive}(\theta, \phi)$, where $\lambda > 0$ measures how much contractive-ness we want to enforce
- The contractive regularization loss itself is defined as the expected Frobenius norm of the Jacobian matrix of the encoder activations with respect to the input: $L_{contractive}(\theta, \phi) = \mathbb{E}_{x \sim \mu_{ref}} ||\nabla_x \mathcal{E}_\phi(x)||_F^2$
- To understand what $L_{contractive}$ measures, note the fact $||\mathcal{E}_\phi(x + \delta x) - \mathcal{E}_\phi(x)||_2 \leq ||\nabla_x \mathcal{E}_\phi(x)||_F ||\delta x||_2$ for any message $x \in \mathcal{X}$, and small variation δx in it.

Contractive Autoencoders (CAE)

- A contractive autoencoder adds the contractive regularization loss to the standard autoencoder loss: $\min_{\theta,\phi} L(\theta, \phi) + \lambda L_{contractive}(\theta, \phi)$, where $\lambda > 0$ measures how much contractive-ness we want to enforce
- The contractive regularization loss itself is defined as the expected Frobenius norm of the Jacobian matrix of the encoder activations with respect to the input: $L_{contractive}(\theta, \phi) = \mathbb{E}_{x \sim \mu_{ref}} ||\nabla_x \mathcal{E}_\phi(x)||_F^2$
- To understand what $L_{contractive}$ measures, note the fact $||\mathcal{E}_\phi(x + \delta x) - \mathcal{E}_\phi(x)||_2 \leq ||\nabla_x \mathcal{E}_\phi(x)||_F ||\delta x||_2$ for any message $x \in \mathcal{X}$, and small variation δx in it.
- Thus, if $||\nabla_x \mathcal{E}_\phi(x)||_F^2$ is small, it means that a small neighborhood of the message maps to a small neighborhood of its code. This is a desired property, as it means small variation in the message leads to small, perhaps even zero, variation in its code, like how two pictures may look the same even if they are not exactly the same

Table of Contents

What are Variational Auto Encoders?

- VAEs are called Auto Encoders only because of architectural similarity,they differ significantly in their mathematical analysis and final goals

What are Variational Auto Encoders?

- VAEs are called Auto Encoders only because of architectural similarity,they differ significantly in their mathematical analysis and final goals
- Allow modelling a **statistical inference** problem as a **statistical optimization** problem

What are Variational Auto Encoders?

- VAEs are called Auto Encoders only because of architectural similarity,they differ significantly in their mathematical analysis and final goals
- Allow modelling a **statistical inference** problem as a **statistical optimization** problem
- Originally for unsupervised data, though recently its effectiveness has been proven for semi-supervised and supervised versions too

What are Variational Auto Encoders?

- VAEs are called Auto Encoders only because of architectural similarity,they differ significantly in their mathematical analysis and final goals
- Allow modelling a **statistical inference** problem as a **statistical optimization** problem
- Originally for unsupervised data, though recently its effectiveness has been proven for semi-supervised and supervised versions too
- The input data is essentially sampled from an (unkown) parametrized distribution, which needs to be modelled

What are Variational Auto Encoders?

- VAEs are called Auto Encoders only because of architectural similarity,they differ significantly in their mathematical analysis and final goals
- Allow modelling a **statistical inference** problem as a **statistical optimization** problem
- Originally for unsupervised data, though recently its effectiveness has been proven for semi-supervised and supervised versions too
- The input data is essentially sampled from an (unkown) parametrized distribution, which needs to be modelled
- The encoder and decoder are jointly trained to minimize a reconstruction error (usually in the sense of the **Kullback - Leibler divergence**)

Mathematical Formulation

- The objective is to model the data's true distribution P using parametrized distribution $p_\theta(x)$

Mathematical Formulation

- The objective is to model the data's true distribution P using parametrized distribution $p_\theta(x)$
- Let z be a random variable jointly distributed with x. Taking the marginal, we obtain

$$p_\theta(x) = \int_z p_\theta(x|z)p_\theta(z)dz$$

Mathematical Formulation

- The objective is to model the data's true distribution P using parametrized distribution $p_\theta(x)$
- Let z be a random variable jointly distributed with x. Taking the marginal, we obtain

$$p_\theta(x) = \int_z p_\theta(x|z)p_\theta(z)dz$$

- A trivil example is the construction of a Guassian Mixture model using $p_\theta(x|z)$ to be a Gaussian, which lets us notice the prior $p_\theta(z)$, likelihood $p_\theta(x|z)$ and posterior $p_\theta(z|x)$ terms

Mathematical Formulation

- The objective is to model the data's true distribution P using parametrized distribution $p_\theta(x)$
- Let z be a random variable jointly distributed with x. Taking the marginal, we obtain
$$p_\theta(x) = \int_z p_\theta(x|z)p_\theta(z)dz$$
- A trivil example is the construction of a Guassian Mixture model using $p_\theta(x|z)$ to be a Gaussian, which lets us notice the prior $p_\theta(z)$, likelihood $p_\theta(x|z)$ and posterior $p_\theta(z|x)$ terms
- Computation of $p_\theta(x)$ is often intractable, making us further aproximate it using another function
$$q_\phi(z|x) \approx p_\theta(z|x)$$

Mathematical Formulation

- The objective is to model the data's true distribution P using parametrized distribution $p_\theta(x)$
- Let z be a random variable jointly distributed with x. Taking the marginal, we obtain

$$p_\theta(x) = \int_z p_\theta(x|z)p_\theta(z)dz$$

- A trivil example is the construction of a Guassian Mixture model using $p_\theta(x|z)$ to be a Gaussian, which lets us notice the prior $p_\theta(z)$, likelihood $p_\theta(x|z)$ and posterior $p_\theta(z|x)$ terms
- Computation of $p_\theta(x)$ is often intractable, making us further aproximate it using another function

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

- Considering z as a latent encoding, we can understand $p_\theta(x|z)$ as a decoder and $q_\phi(z|x)$ as an encoder

Table of Contents

Defining ELBO and its meaning

- As the idea in VAEs is to optimize θ and ϕ to reduce the reconstruction error, we choose the loss to be the Kullback - Leibler divergence

Defining ELBO and its meaning

- As the idea in VAEs is to optimize θ and ϕ to reduce the reconstruction error, we choose the loss to be the Kullback - Leibler divergence

Differentiable Loss Function

$$D_{KL}(q_\phi(z|x)||p_\theta(z|x)) = \mathbb{E}_{z\sim q_\phi(.|x)}[\ln \frac{q_\phi(z|x)}{p_\theta(z|x)}] = \ln p_\theta(x) + \mathbb{E}_{z\sim q_\phi(.|x)}[\ln \frac{q_\phi(z|x)}{p_\theta(x,z)}] \quad (1)$$

- We need a differetiable loss function to lett back-propogation work

Defining ELBO and its meaning

- As the idea in VAEs is to optimize θ and ϕ to reduce the reconstruction error, we choose the loss to be the Kullback - Leibler divergence

Differentiable Loss Function

$$D_{KL}(q_\phi(z|x)||p_\theta(z|x)) = \mathbb{E}_{z\sim q_\phi(.|x)}[\ln \frac{q_\phi(z|x)}{p_\theta(z|x)}] = \ln p_\theta(x) + \mathbb{E}_{z\sim q_\phi(.|x)}[\ln \frac{q_\phi(z|x)}{p_\theta(x,z)}] \tag{1}$$

- We need a differetiable loss function to lett back-propogation work

ELBO

$$L_{\theta,\phi}(x) := \mathbb{E}_{z\sim q_\phi(.|x)}[\ln \frac{p_\theta(x,z)}{q_\phi(z|x)}] = \ln p_\theta(x) - D_{KL}(q_\phi(z|x)||p_\theta(z|x)) \tag{2}$$

- which is maximized to maximize $\ln p_\theta$ and minimize loss

Reparametrization

- To execute gradient descent, we find the gradients with respect to θ and ϕ. The gradient for θ is simply $\mathbb{E}_{z \sim q_\phi(.|x)}[\nabla_\theta \ln \frac{p_\theta(x,z)}{q_\phi(z|x)}]$

Reparametrization

- To execute gradient descent, we find the gradients with respect to θ and ϕ. The gradient for θ is simply $\mathbb{E}_{z \sim q_\phi(.|x)}[\nabla_\theta \ln \frac{p_\theta(x,z)}{q_\phi(z|x)}]$
- The gradient with respect to ϕ is not so easy to calculate, so we need to use a trick known as reparametrization. A simple example is that if $z \sim q_\phi(.|x)$ is normally distributed, we can reparamterize it to $z = \mu_\phi(x) + L_\phi(x)\epsilon$, where $L_\phi(x)$ is the Cholesky decomposition of the Covariance matrix

Reparametrization

- To execute gradient descent, we find the gradients with respect to θ and ϕ. The gradient for θ is simply $\mathbb{E}_{z\sim q_\phi(.|x)}[\nabla_\theta \ln \frac{p_\theta(x,z)}{q_\phi(z|x)}]$
- The gradient with respect to ϕ is not so easy to calculate, so we need to use a trick known as reparametrization. A simple example is that if $z \sim q_\phi(.|x)$ is normally distributed, we can reparamterize it to $z = \mu_\phi(x) + L_\phi(x)\epsilon$, where $L_\phi(x)$ is the Cholesky decomposition of the Covariance matrix
- We can then obtain $\ln q_\phi(z|x) = \ln q_0(\epsilon) - \ln|\det(\partial_\epsilon z)|$ for any general reparametrization of z into ϵ

Thanks for your attention! Any questions?

Hope you slept comfortably!