

Autoencoders

Santiago VELASCO-FORERO
<http://cmm.ensmp.fr/velasco/>

MINES ParisTech
PSL Research University
Center for Mathematical Morphology



Contents

1 Introduction

2 Autoencoder

- Autoencoder
- Type of Autoencoder
- Variational Autoencoder

3 Applications for Autoencoders

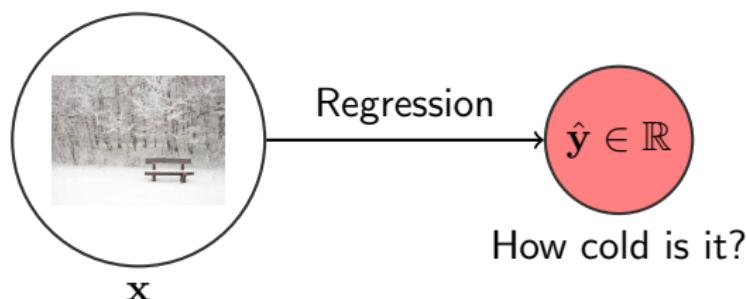
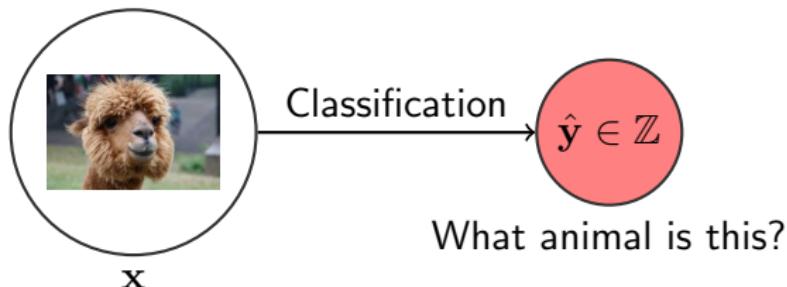
4 References

Contents

- 1 Introduction
- 2 Autoencoder
- 3 Applications for Autoencoders
- 4 References

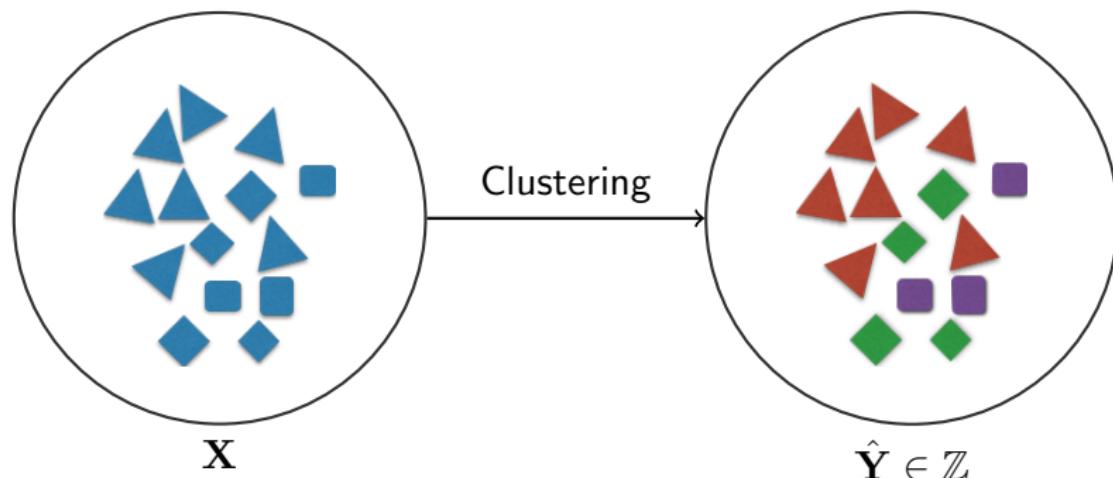
Supervised Learning

Given a labeled dataset (\mathbf{X}, \mathbf{Y}) , we would like to learn a mapping from data space to label space.



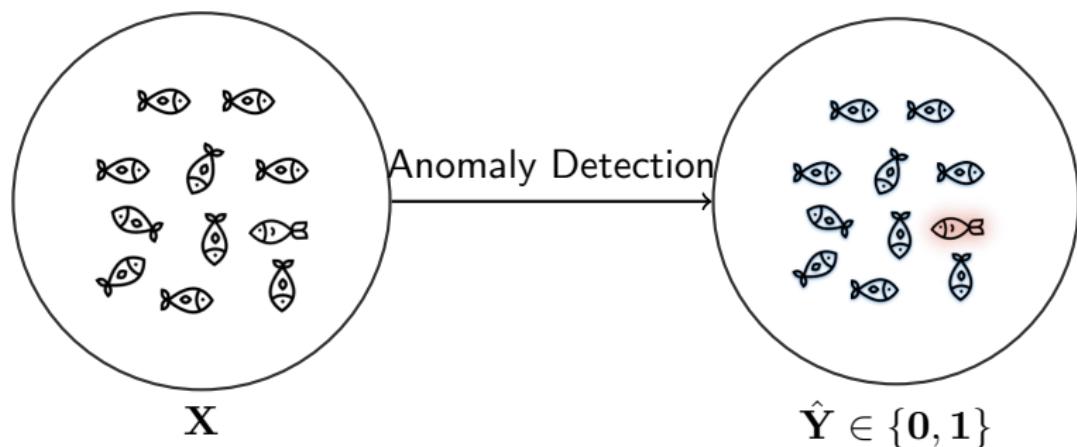
Unsupervised Learning: Clustering

Given an unlabeled dataset (\mathbf{X}), we would like to learn: **How to group objects into similar categories?**



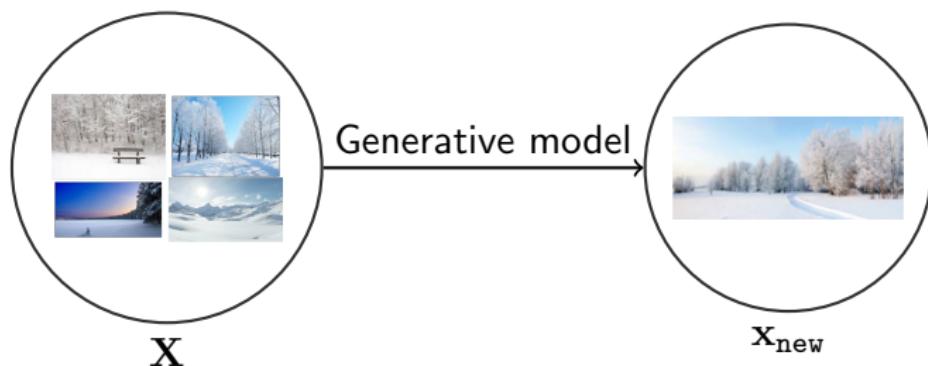
Unsupervised Learning: Anomaly detection

Given an unlabeled dataset (\mathbf{X}), we would like to learn: How to identify observations differing significantly from the majority of data?



Unsupervised learning: Generative Models

Given an unlabeled dataset (\mathbf{X}), we would like to learn: How to generate a new observations from the same distribution (unknown) of dataset?



Autoencoders

Autoencoders are neural networks whose purpose is twofold.

- ① To compress some input data by transforming it from the input domain to another space, known as the *latent space* (code).
- ② To take this latent representation and transform it back to the original space, such that the output is similar to the input.

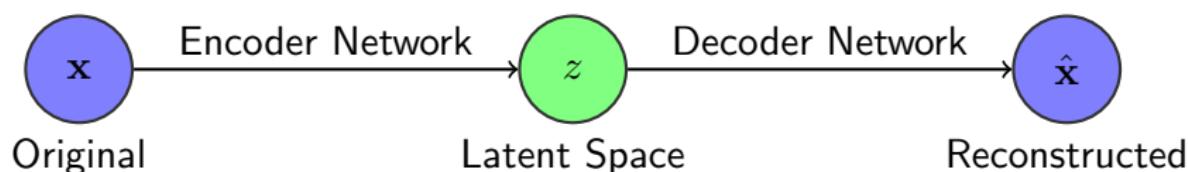


Figure: Loss function for a given input vector is usually their reconstruction error, i.e., $L(\mathbf{x}) = (\mathbf{x} - \hat{\mathbf{x}})^2$

Contents

1 Introduction

2 Autoencoder

- Autoencoder
- Type of Autoencoder
- Variational Autoencoder

3 Applications for Autoencoders

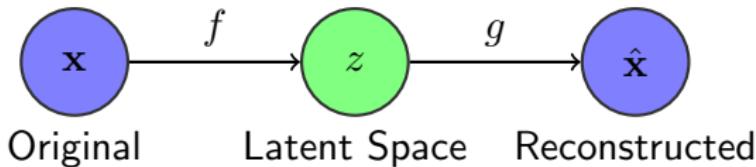
4 References

Autoencoder

- An *autoencoder* is a neural network that is trained to attempt to copy its input to its output.
- The network may be viewed as consisting of two parts: an encoder function $h = f(\mathbf{X})$ and a decoder that produces a reconstruction $r(\mathbf{X}) = g(h(\mathbf{X}))$.
- The composition of f and g is called the *reconstruction function*
- If an autoencoder succeeds to learn $g(f(\mathbf{X})) = \mathbf{X}$ everywhere, then it is not especially useful (overfitting).
- The learning process can be described as minimizing the loss function:

$$L(\mathbf{X}, g(f(\mathbf{X}))), \quad (1)$$

where L is a loss function, such as mean squared error.



Over/Under complete autoencoders

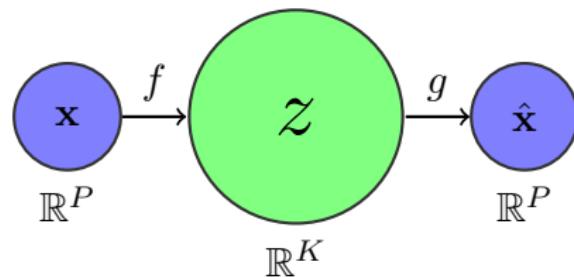


Figure: $P < K$: Overcomplete AE

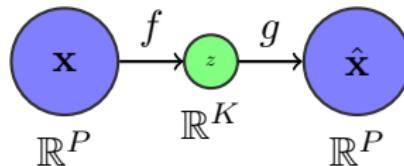


Figure: $K < P$: Undercomplete AE

An autoencoder whose code dimension is less than the input dimension is called **undercomplete**.

Contents

1 Introduction

2 Autoencoder

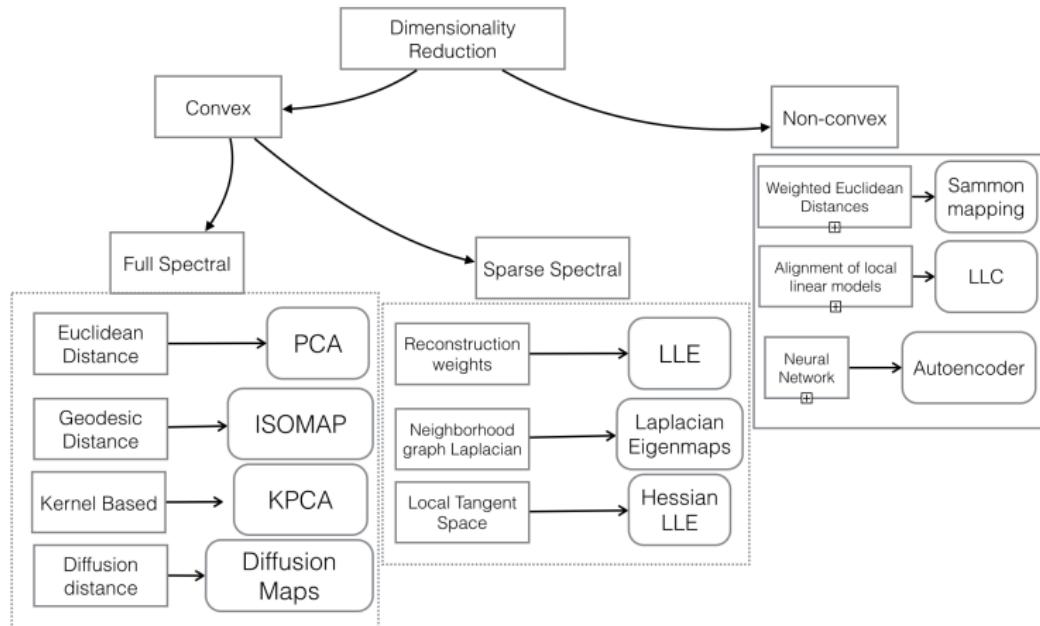
- Autoencoder
- Type of Autoencoder
- Variational Autoencoder

3 Applications for Autoencoders

4 References

Dimensional reduction methods

When the decoder is linear and L is the mean squared error, an undercomplete autoencoder learns to span the same subspace as PCA.



Autoencoder vs general data compression methods

- Autoencoder are data-dependent
- MP3 or JPEG compression algorithm make general assumptions about "sound/images?", but not about specific types of sounds/image.
- Autoencoders are lossy.
- Autoencoders are learnt for a specific application.

Motivation: Nonlinear dimensionality reduction

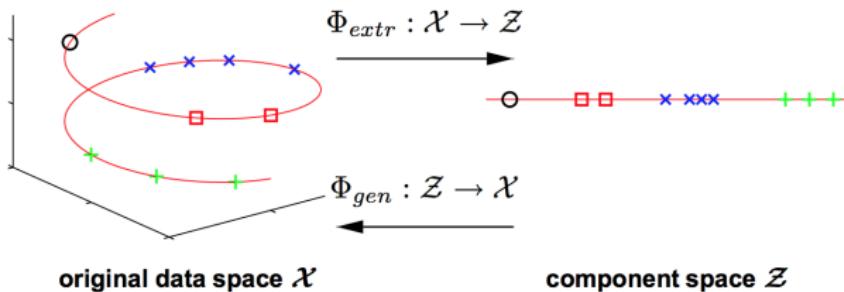


Figure: TODO: Manifold Learning

Contents

1 Introduction

2 Autoencoder

- Autoencoder
- Type of Autoencoder
- Variational Autoencoder

3 Applications for Autoencoders

4 References

Type of Autoencoders:

- ① Vanilla autoencoder
- ② Regularized autoencoder (Sparse)
- ③ Denoising autoencoder
- ④ Contractive autoencoder
- ⑤ Variational autoencoder

Vanilla (Standard) Autoencoder

```
dim_input=100
dim_latent_space=3
dim_layers=[50,10]
#Encoder
input_img = Input(shape=(dim_input,))
encoded1 = Dense(dim_layers[0], activation='relu',name='encoded1')(input_img)
encoded2 = Dense(dim_layers[1], activation='relu',name='encoded2')(encoded1)
z = Dense(dim_latent_space, activation='relu',name='latent')(encoded2)
encoder = Model(input_img, z)

#Decoder
decoded1 = Dense(dim_layers[1], activation='relu',name='decoded1')(y)
decoded2 = Dense(dim_layers[0], activation='relu',name='decoded2')(decoded1)
y = Dense(dim_input, activation='relu',name='z')(decoded2)
autoencoder = Model(input_img, y)

autoencoder.compile(optimizer='adam', loss='mse')
autoencoder.summary()
```

Figure: Vanilla Autoencoder. Poor initialization can lead to local minima.

- Rumelhart, Hinton, Williams [RHW88]: Random Initialization and gradient descent shows bad performance.
- Bengio, Lamblin, Popovici, Larochelle [BLP+07] [PCL06]: Stacking Autoencoders and tune with gradient descent shows good performance.

Regularized Autoencoders

A sparse autoencoder is simply an autoencoder whose training criterion involves a sparsity penalty $\Omega(h)$ on the code layer h , in addition to the reconstruction error:

$$L(\mathbf{X}, g(f(\mathbf{X}))) + \Omega(h) \quad (2)$$

- L_1 : Cost function = Loss Function $+ \frac{\lambda}{2m} \sum ||w||$
- L_2 Cost function = Loss Function $+ \frac{\lambda}{2m} \sum ||w||^2$

```
from keras import regularizers
model.add(Dense(64, input_dim=64,
                kernel_regularizer=regularizers.l2(0.01))
```

Figure: Regularizers in Keras. Note: Kernel and bias regularizer are not the same.

Sparse Autoencoders

Regularization of the representation learned by the Auto-Encoders.

- Enforcing most code coefficients to be close to 0 (to be inactive).
- Capturing a more robust representation of the manifold structure.

Common implementation

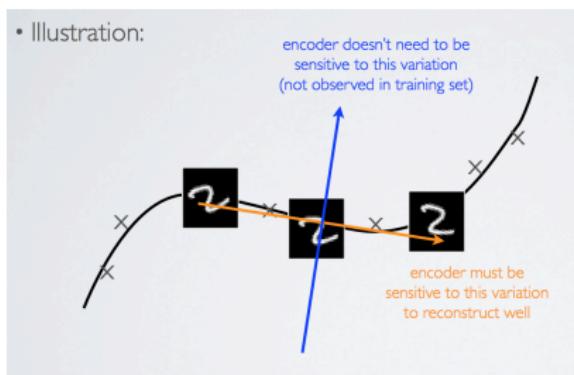
- Adding a sparsity regularizer loss to the autoencoder loss function.
- Various sparsity regularizers.
- Other existing methods:

Contractive Autoencoders

The *contractive autoencoder* introduces a regularizer on the code $h = f(\mathbf{X})$, encouraging the derivatives of f to be as small as possible:

$$\Omega(h) = \lambda \left\| \frac{\partial f(\mathbf{X})}{\partial \mathbf{X}} \right\|_F^2 \quad (3)$$

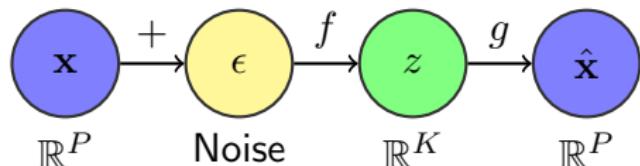
TODO



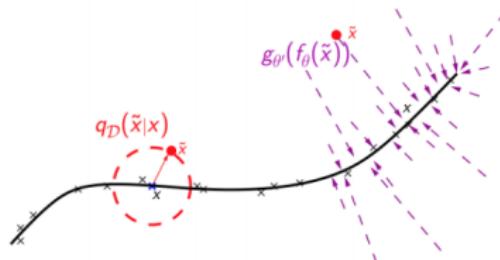
Denoising Autoencoders (DAE)

$$L(\mathbf{X}, g(f(\tilde{\mathbf{X}}))), \quad (4)$$

where $\tilde{\mathbf{X}}$ is a corrupted copy of \mathbf{X} by some form of noise. An overcomplete autoencoder with high capacity can end up learning an identity function where input=output. Add noise to



Interpretation: Manifold Learning



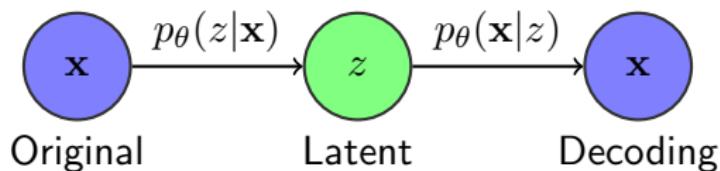
- training data lies nearby a low-dimensional manifold.
- a corrupted example is obtained by applying a perturbation of original example
- The model should learn to *project them back* to the manifold

TODO

Modern Autoencoder

- Modern autoencoders have generalized the idea of an encoder and a decoder beyond deterministic functions to stochastic mappings $p_{\text{encoder}}(h|\mathbf{X})$ and $p_{\text{decoder}}(\mathbf{X}|h)$.

Variational Autoencoder



- Training via maximum likelihood of $p(\mathbf{x})$
- Intractability: the true posterior density $p_\theta(z|\mathbf{x})$ can be calculated
- Solutions: a) MCMC (too costly) b) Approximate $p(z|\mathbf{x})$ by means of $q(z|\mathbf{x}) = \mathcal{N}(z; \mu_z(x), \sigma_z(x))$

Contents

1 Introduction

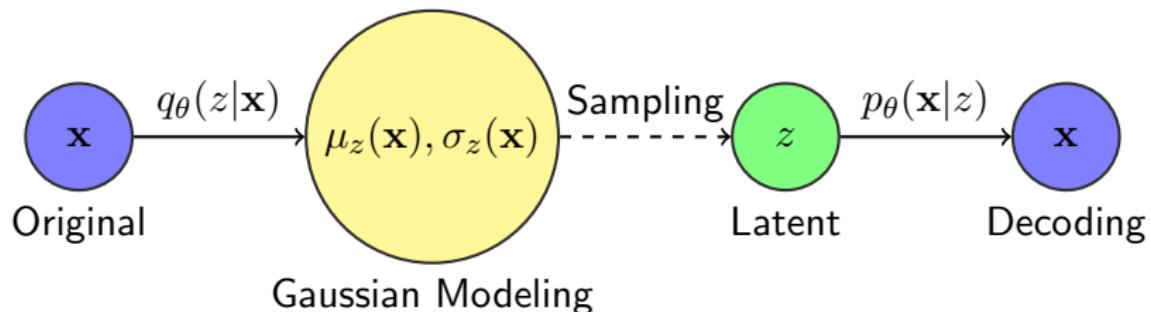
2 Autoencoder

- Autoencoder
- Type of Autoencoder
- Variational Autoencoder

3 Applications for Autoencoders

4 References

Variational Autoencoder



- Gaussian modeling (diagonal covariance matrix to avoid problems in high dimensionality)
- Training via maximum likelihood of $p(x)$
- Learning the parameters θ 's via backpropagation?

Training via maximum likelihood

Assume we would like to compute the likelihood of an image \mathbf{x} from the training set:

$$\begin{aligned}\mathcal{L}(\mathbf{x}) &= \log(p(\mathbf{x})) \\ &= \sum_z q(z|\mathbf{x}) \log(p(\mathbf{x})) \\ &= \sum_z q(z|\mathbf{x}) \log\left(\frac{p(z, \mathbf{x})}{p(z|\mathbf{x})}\right) \\ &= \sum_z q(z|\mathbf{x}) \log\left(\frac{p(z, \mathbf{x})q(z|\mathbf{x})}{q(z|\mathbf{x})p(z|\mathbf{x})}\right) \\ &= \sum_z q(z|\mathbf{x}) \log\left(\frac{p(z, \mathbf{x})}{q(z|\mathbf{x})}\right) + \sum_z q(z|\mathbf{x}) \log\left(\frac{q(z|\mathbf{x})}{p(z|\mathbf{x})}\right) \\ &= \sum_z q(z|\mathbf{x}) \log\left(\frac{p(z, \mathbf{x})}{q(z|\mathbf{x})}\right) + \underbrace{D_{KL}(q(z|\mathbf{x}), p(z|\mathbf{x}))}_{\text{how good is the approximation.}} \\ &= \mathcal{L}^{lrb}(\mathbf{x}) + D_{KL}(q(z|\mathbf{x}), p(z|\mathbf{x})) \\ &\geq \mathcal{L}^{lrb}(\mathbf{x})\end{aligned}$$

TODO CHECK

$$\begin{aligned}\mathcal{L}(\mathbf{x}) \geq \mathcal{L}^{lvb}(\mathbf{x}) &= \sum_z q(z|\mathbf{x}) \log \left(\frac{p(z, \mathbf{x})}{q(z|\mathbf{x})} \right) \\&= \sum_z q(z|\mathbf{x}) \log \left(\frac{p(\mathbf{x}|z)p(z)}{q(z|\mathbf{x})} \right) \\&= \sum_z q(z|\mathbf{x}) \log \left(\frac{p(\mathbf{x}|z)}{q(z|\mathbf{x})} \right) + \sum_z q(z|\mathbf{x}) \log \left(\frac{p(z)}{q(z|\mathbf{x})} \right) \\&= \mathbb{E}_{q(z|\mathbf{x})} \log(p(\mathbf{x}|z)) - D_{KL}(q(z|\mathbf{x}), p(z)) \\&= \underbrace{\mathbb{E}_{q(z|\mathbf{x})} \log(p(\mathbf{x}|z))}_{\text{Expected Reconstruction}} - \underbrace{D_{KL}(q(z|\mathbf{x}), p(z))}_{\text{Regularization } \mathcal{N}(0, 1)}\end{aligned}$$

- First term implies the use of many realization of sampling process (in practice we only have a few samples per training example!)
- Second term is simply a formula for diagonal multivariate Gaussian distribution.

Reparametrization trick

Backpropagation is not possible through random sampling!

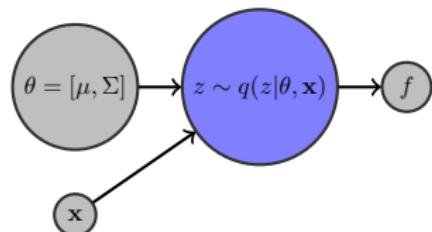


Figure: Original Formulation

Reparametrization trick

Backpropagation is not possible through random sampling!

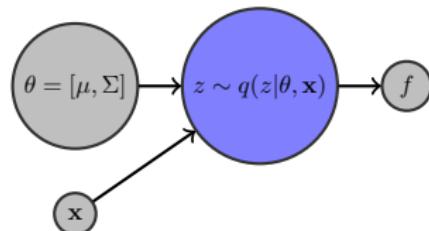


Figure: Original Formulation

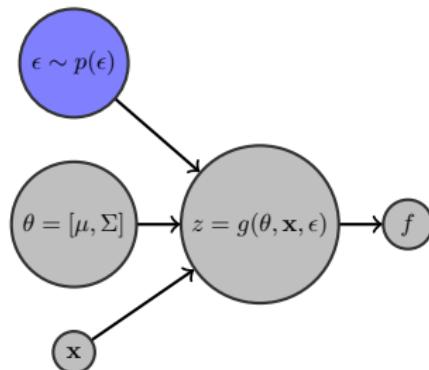


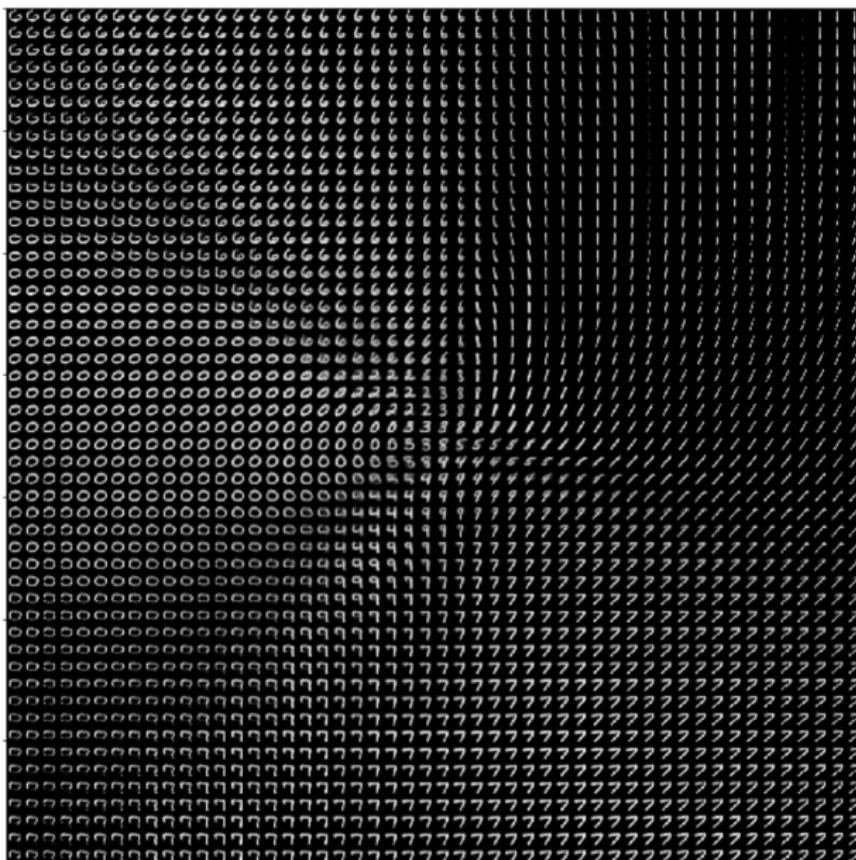
Figure: Reparametrization trick. Backpropagation

Contents

- 1 Introduction
- 2 Autoencoder
- 3 Applications for Autoencoders
- 4 References

- Data denosing
- Dimensionality Reduction
- Database understanding
- Generative models

Example Latent Space for MNIST



Applications for Autoencoders

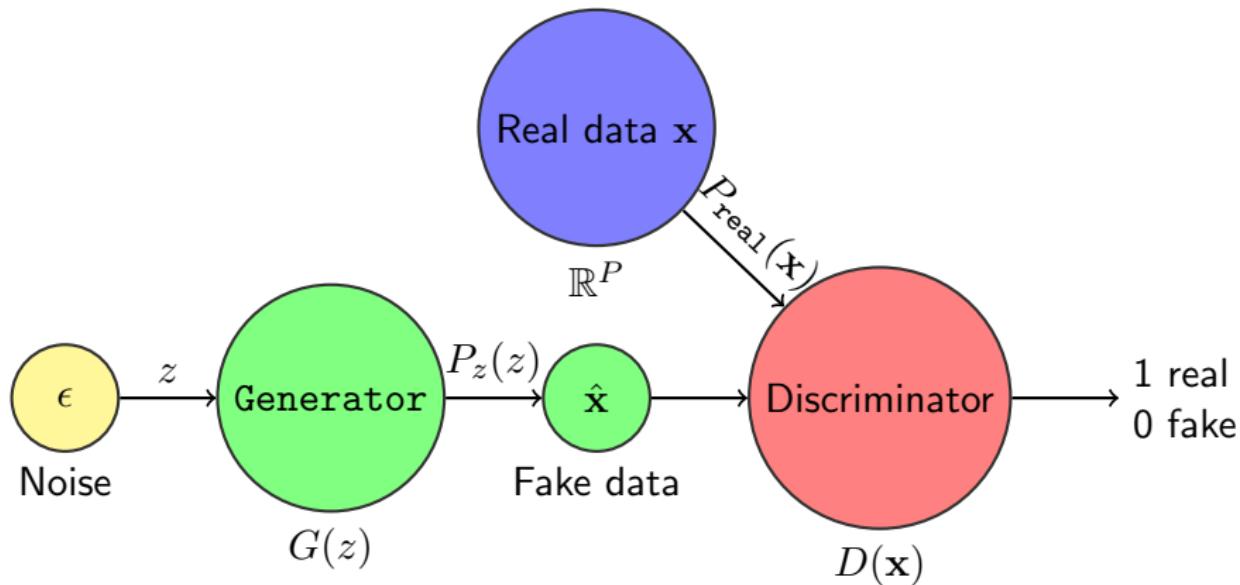
- ① Information retrieval using deep auto-encoders (semantic hashing)

KL-divergence between two multivariate Gaussian distributions

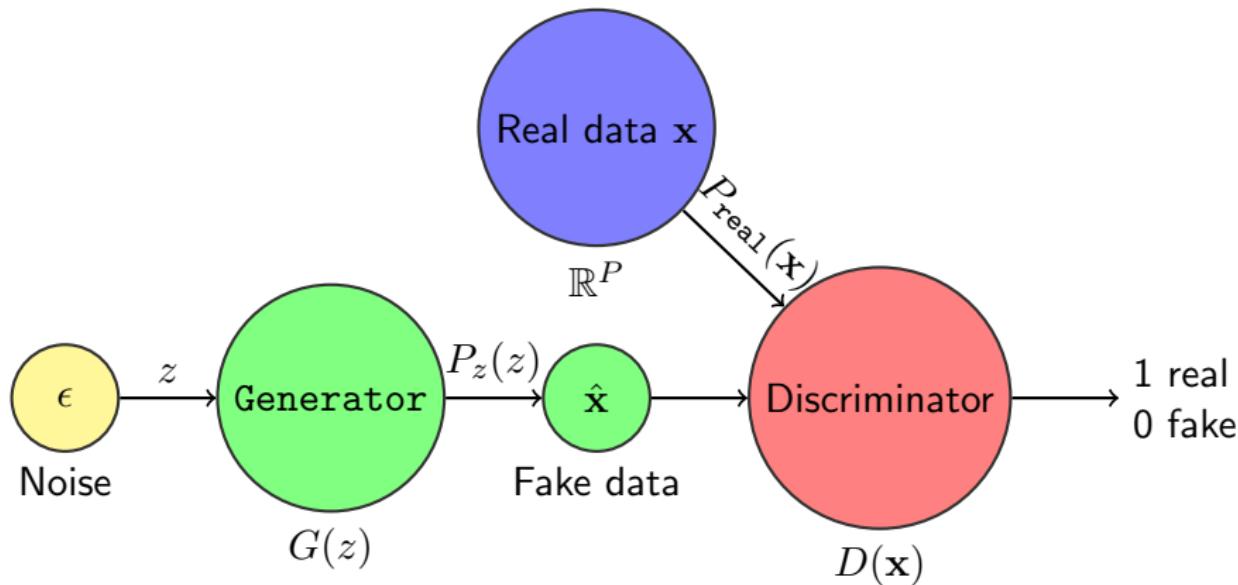
$$KL(\mathcal{N}(\mu_0, \Sigma_0))||\mathcal{N}(\mu_1, \Sigma_1)) = \frac{1}{2} \left(\text{tr}(\Sigma_1^{-1} \Sigma_0) + (\mu_1 - \mu_0)^T \Sigma_1^{-1} (\mu_1 - \mu_0) - k + \log \left(\frac{\det \Sigma_1}{\det \Sigma_0} \right) \right)$$

where k is the dimensionality of the distributions.

Generative Adversarial Networks (GANs)



Generative Adversarial Networks (GANs)



The goal is to optimize the following expression,
 $\min_G \max_D V(D, G)$. We are looking for the best generator for the best discriminator.

Generative Adversarial Networks (GANs)

We require that the discriminator D recognizes examples from the $P_{\text{real}}(\mathbf{x})$ distribution,

$$\mathbb{E}_{\mathbf{x} \sim P_{\text{real}}(\mathbf{x})} [\log D(\mathbf{x})] \quad \text{Decision over Real Data}$$

where \mathbb{E} denotes the expectation. This term comes from the “1” class of the log-loss function.

Additionally, we would like to tricking the discriminator via a good generator G . Thus, the term comes from “0” class of the log-loss function:

$$\mathbb{E}_{z \sim P_z(z)} [\log(1 - D(G(z)))] \quad \text{Decision over Fake Data}$$

Optimal Discriminator? Optimal Generator?

We define:

$$V(G, D) := \mathbb{E}_{\mathbf{x} \sim P_{\text{real}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{z \sim P_z(z)} [\log(1 - D(G(z)))]$$

What should be an optimal discriminator?

Optimal Discriminator? Optimal Generator?

We define:

$$V(G, D) := \mathbb{E}_{\mathbf{x} \sim P_{\text{real}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{z \sim P_z(z)} [\log(1 - D(G(z)))]$$

What should be an optimal discriminator?

$$D^* = \arg \max_D V(G, D)$$

Now, given this optimal discriminator D^* , what should be an optimal generator?

Optimal Discriminator? Optimal Generator?

We define:

$$V(G, D) := \mathbb{E}_{\mathbf{x} \sim P_{\text{real}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{z \sim P_z(z)} [\log(1 - D(G(z)))]$$

What should be an optimal discriminator?

$$D^* = \arg \max_D V(G, D)$$

Now, given this optimal discriminator D^* , what should be an optimal generator?

$$G^* = \arg \min_G V(G, D^*)$$

Minimax Game []

$$\begin{aligned}
& \min_G \max_D V(D, G) := \\
&= \mathbb{E}_{\mathbf{x} \sim P_{\text{real}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{z \sim P_z(z)} [\log(1 - D(G(z)))] \\
&\quad \text{by Radon-Nikodym Theorem} \\
&= \mathbb{E}_{\mathbf{x} \sim P_{\text{real}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{x \sim P_{g(\mathbf{x})}} [\log(1 - D(x))]
\end{aligned}$$

An optimal generator should give $P_{g(\mathbf{x})} = P_{\text{real}}(\mathbf{x})$, and in this case:

(5)

Adversarial Example

Given a benign sample \mathbf{x} , an adversarial example \mathbf{x}_{adv} is generated by adding a small perturbation to \mathbf{x} (i.e. $\mathbf{x}_{adv} = \mathbf{x} + \epsilon$), so that \mathbf{x}_{adv} is misclassified by the targeted classifier g .

Formal definition: (Adversarial example (Wang et al., 2016)).

Given a ML model $f(\cdot)$ and a *small perturbation* δ , we call \mathbf{x}_{adv} an adversarial example if there exists x , an example drawn from the benign data distribution, such that $f(\mathbf{x}) \neq f(\mathbf{x}_{adv})$ and $\|\mathbf{x} - \mathbf{x}_{adv}\| \leq \delta$. This is, \mathbf{x}_{adv} could be called a *counter example*.

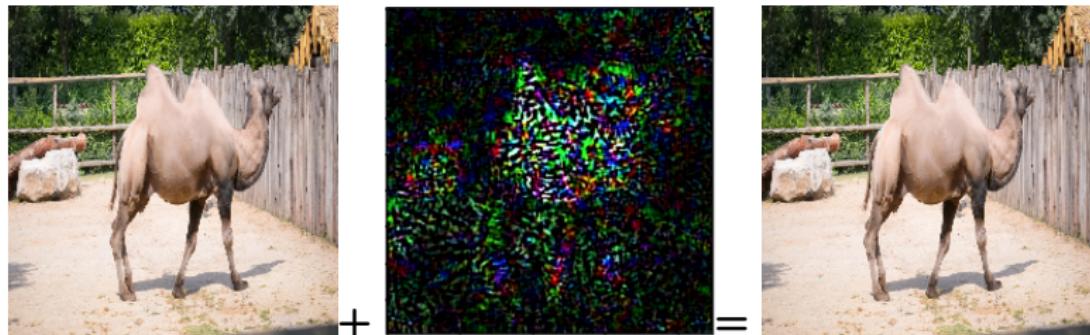


Figure: $\mathbf{x} + \epsilon = \mathbf{x}_{adv}$

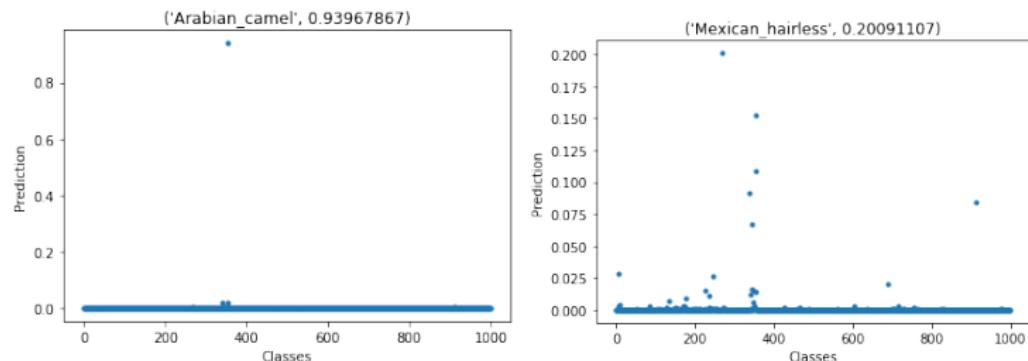


Figure: Prediction for: \mathbf{x} and \mathbf{x}_{adv}

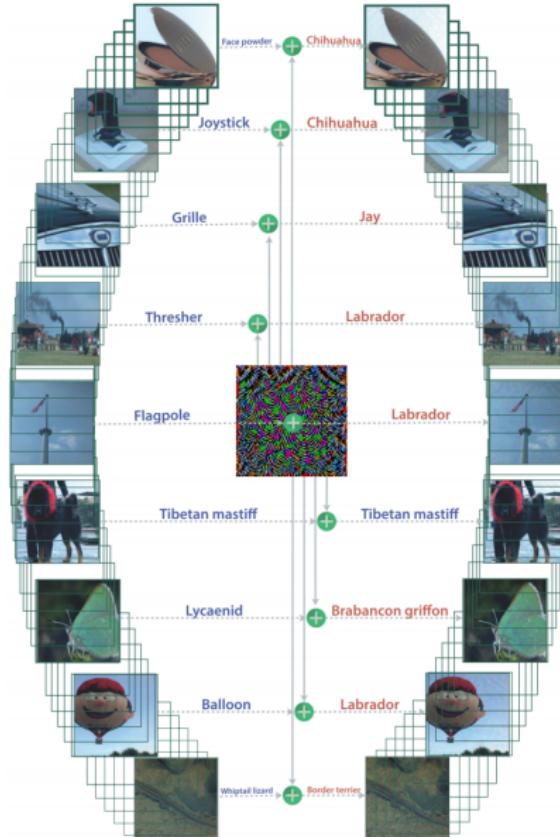
Adversarial Examples

Fast Gradient Sign Method Their perturbation can be expressed as

$$\eta = \epsilon sign(\nabla_{\mathbf{x}} J_{\theta}(\mathbf{x}, L)) \quad (6)$$

where ϵ is the magnitude of the perturbation. The generated adversarial example \mathbf{x}' is calculated as $\mathbf{x}' = \mathbf{x} + \eta$. This perturbation can be computed by using back-propagation.

Universal Perturbation



Contents

1 Introduction

2 Autoencoder

3 Applications for Autoencoders

4 References

- Tutorial on Variational Autoencoders, CARL DOERSCH
 - <http://jmlr.csail.mit.edu/papers/volume15/alain14a/alain14a.pdf>
 - <http://web.stanford.edu/class/cs294a/sae/sparseAutoencoderNotes.pdf>
- Contractive Auto-Encoders: Explicit Invariance During Feature Extraction. Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot et Yoshua Bengio, 2011.