Introduction to Autoencoders, GANs and Adversarial Examples

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

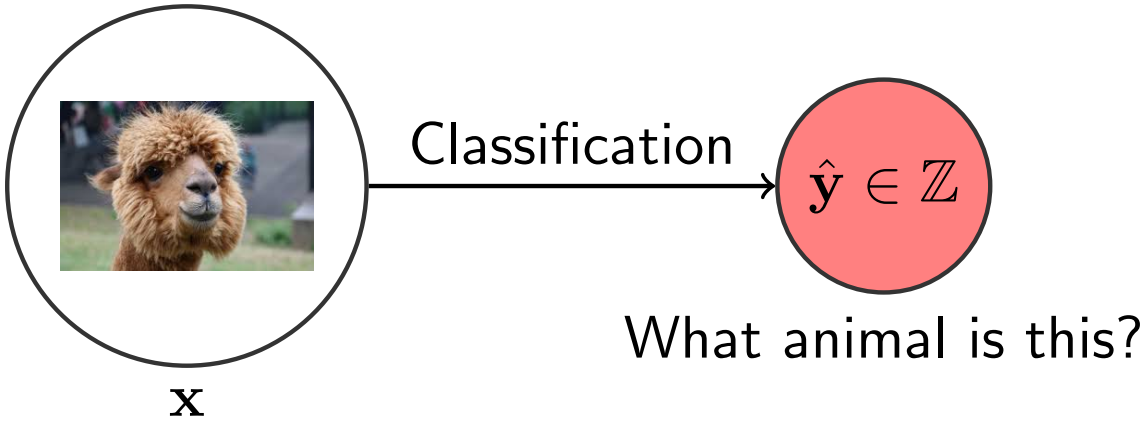
MINES ParisTech
PSL Research University
Center for Mathematical Morphology

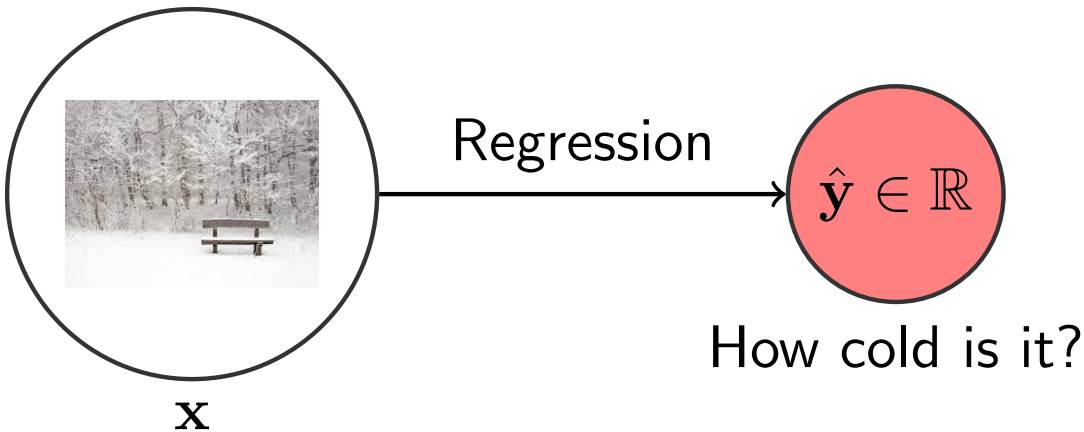
Contents

Contents

Supervised Learning

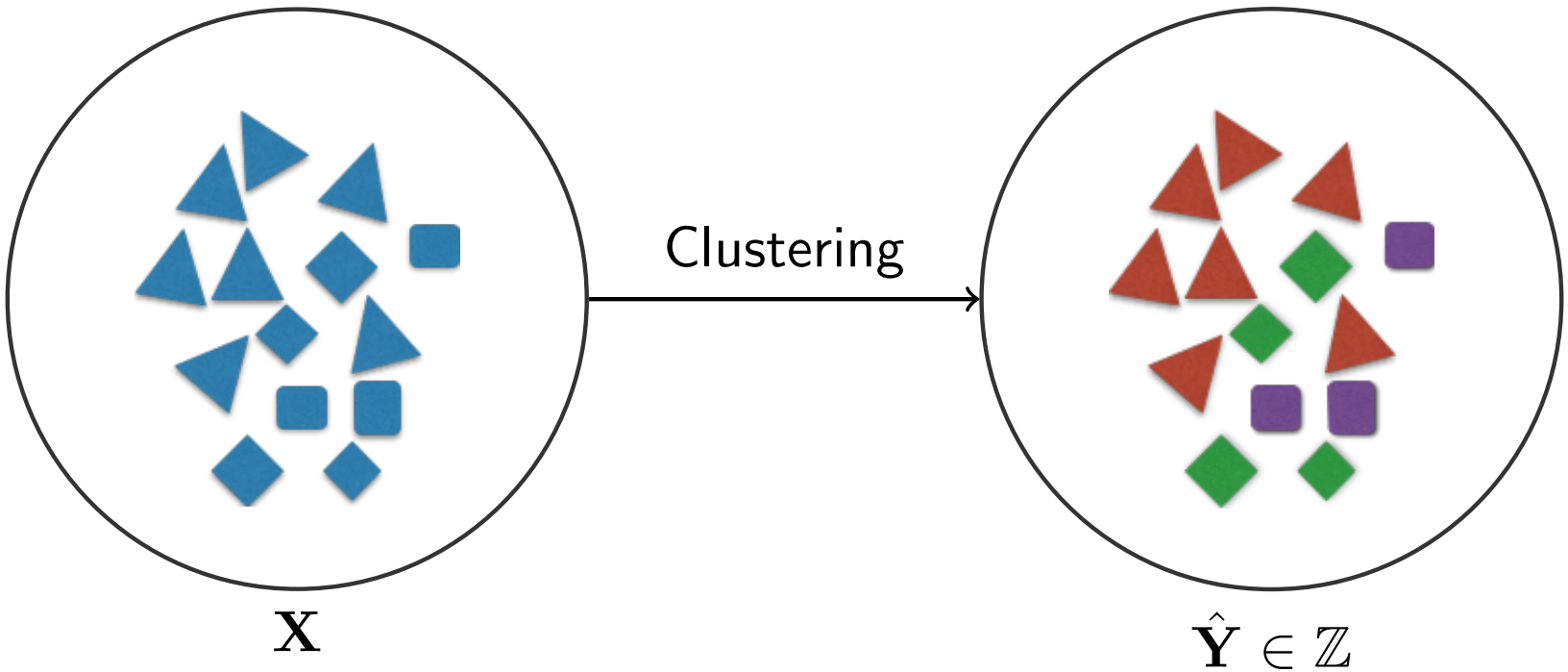
Given a labeled dataset $(\mathbf{X}, \mathbf{Y})$, we would like to learn a mapping from data space to label space.





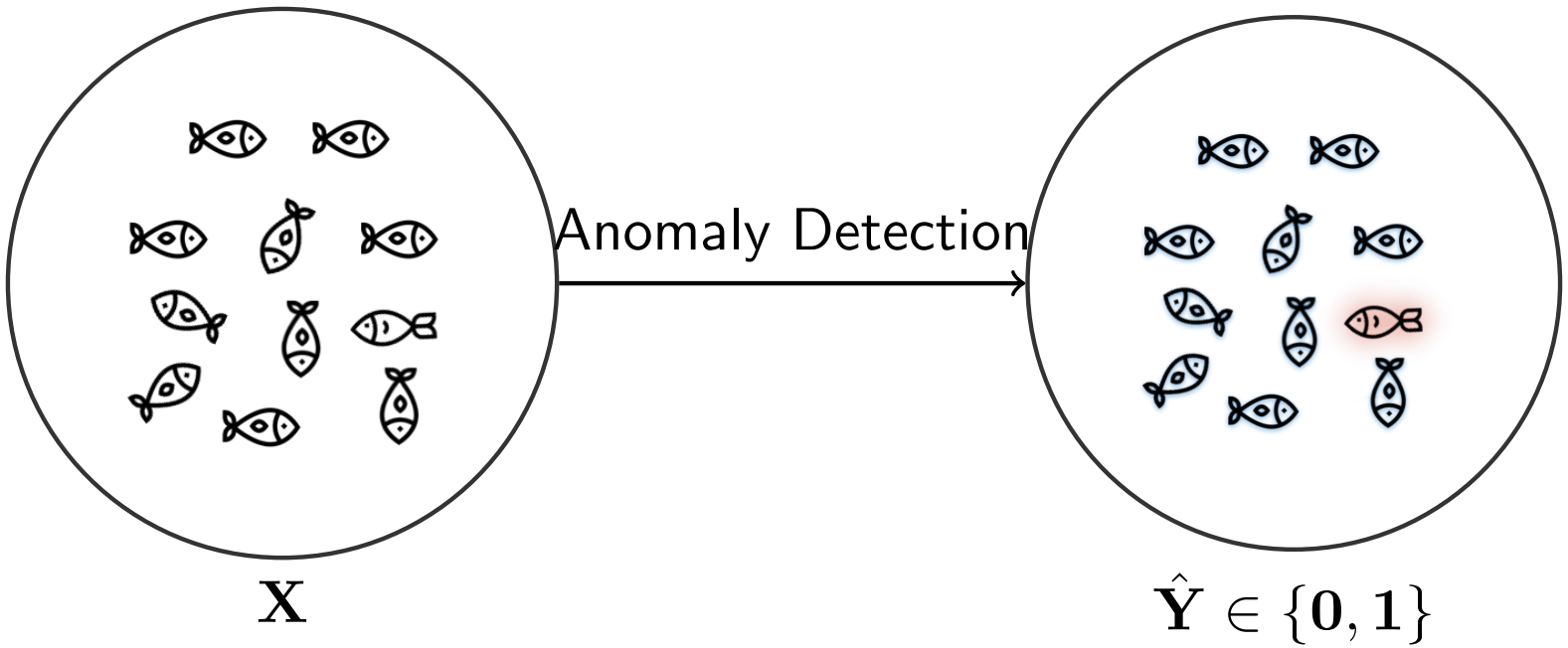
Unsupervised Learning: Clustering

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



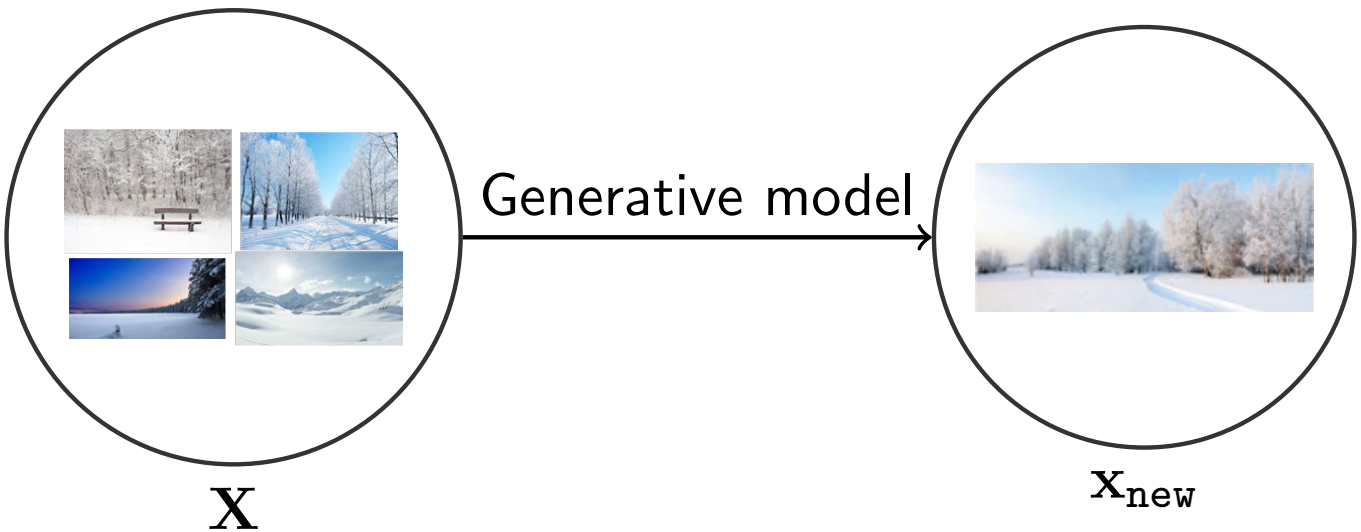
Unsupervised Learning: Anomaly detection

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.

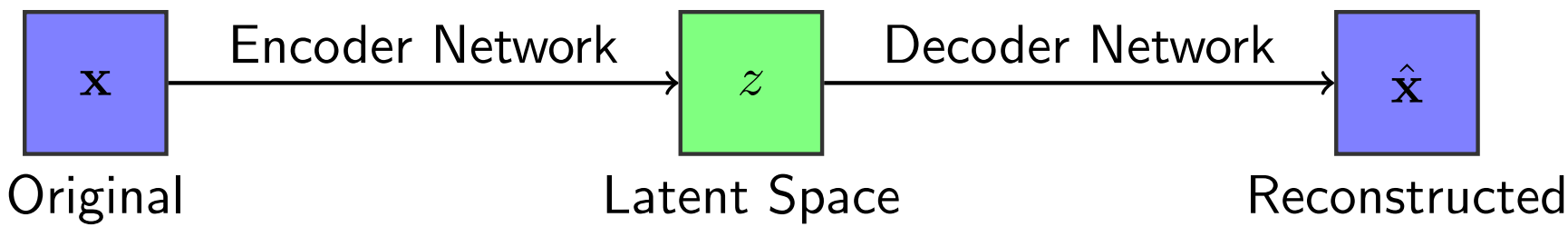


Figure: Loss function for a given input vector is usually their reconstruction error, i.e.,$L(\mathbf{x}) = (\mathbf{x} - \hat{\mathbf{x}})^2$

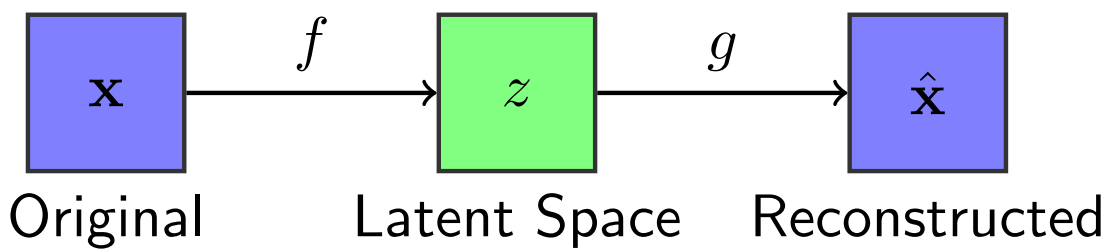
Contents

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:

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

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



Over/Under complete autoencoders

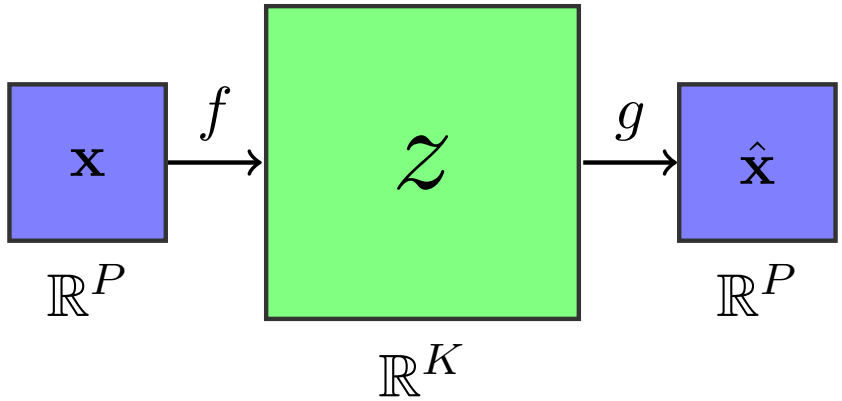


Figure: $P < K$: Overcomplete AE

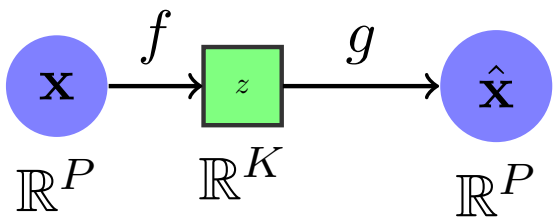
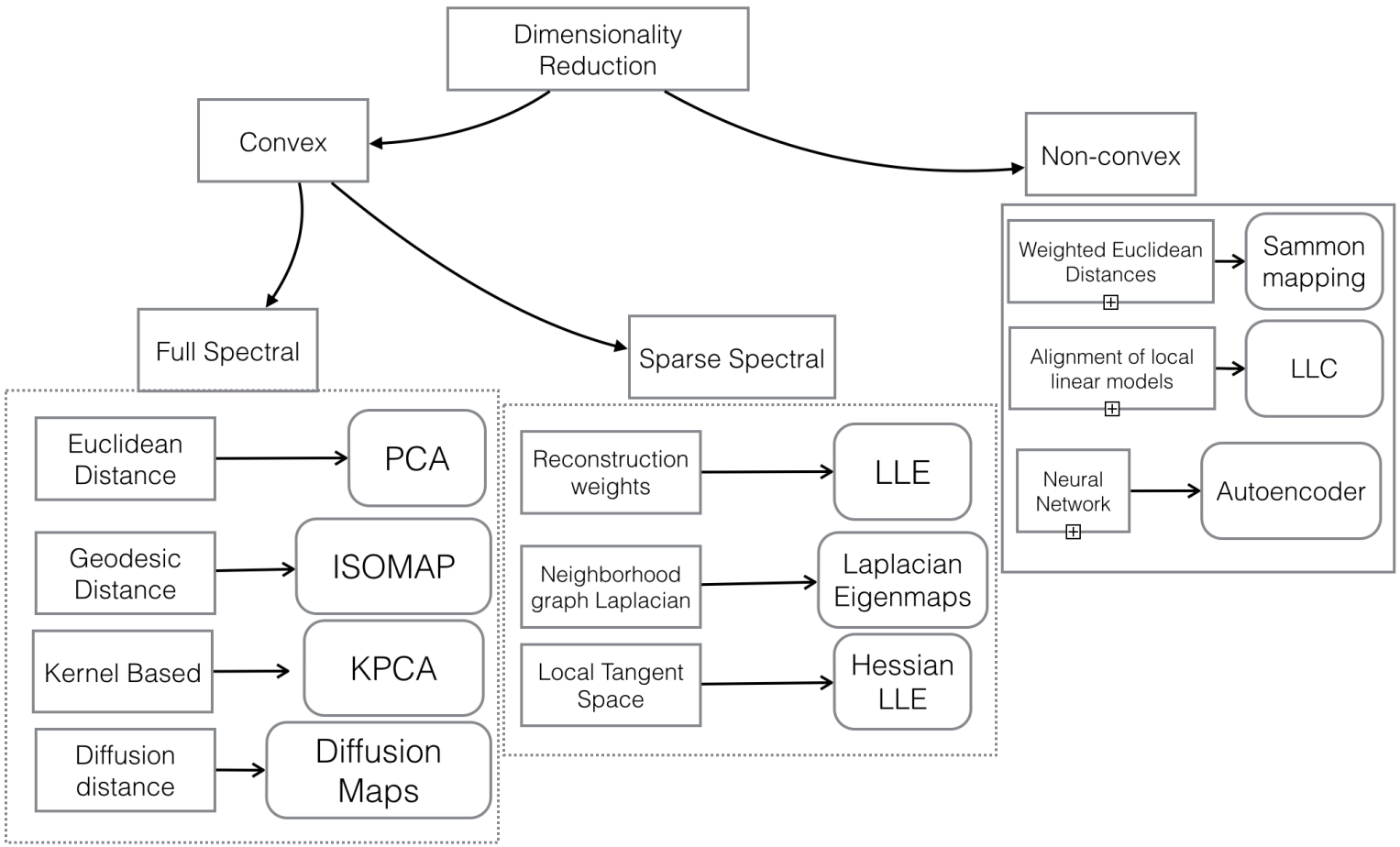


Figure: $K < P$: Undercomplete AE

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

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.



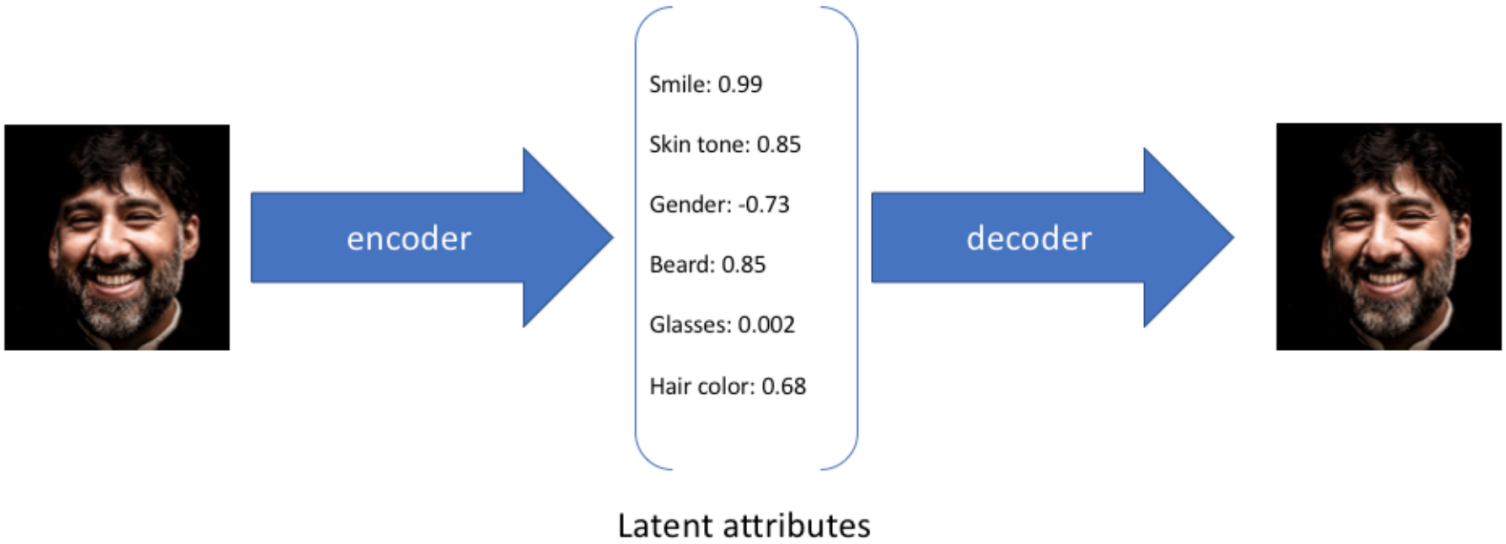


Figure: Latent Space Intuition. Source: `https://www.jeremyjordan.me/variational-autoencoders/`

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.

Contents

Type of Autoencoders:

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

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 et al., 1986]: Random Initialization and gradient descent shows bad performance.
- [Bengio et al., 2007] [Vincent et al., 2010]: 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:

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

- 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.

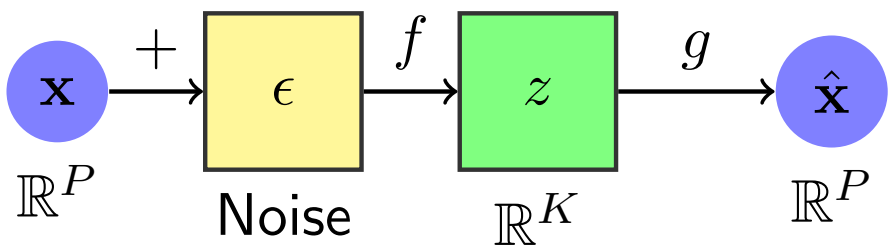
Common implementation

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

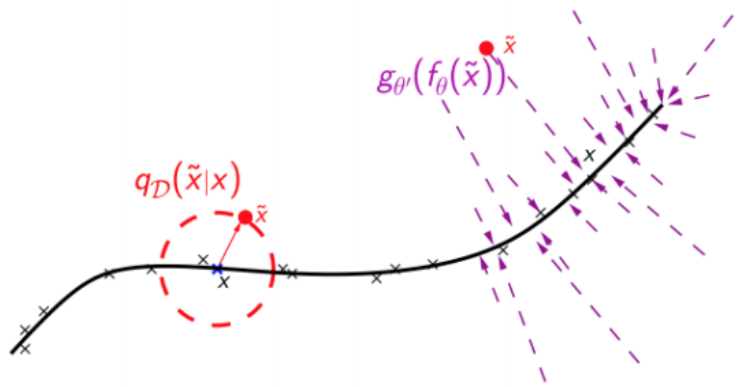
Denoising Autoencoders (DAE)[Vincent et al., 2008]

$$L(\mathbf{X}, g(f(\tilde{\mathbf{X}}))), \tag{3}$$

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



Interpretation: Manifold Learning



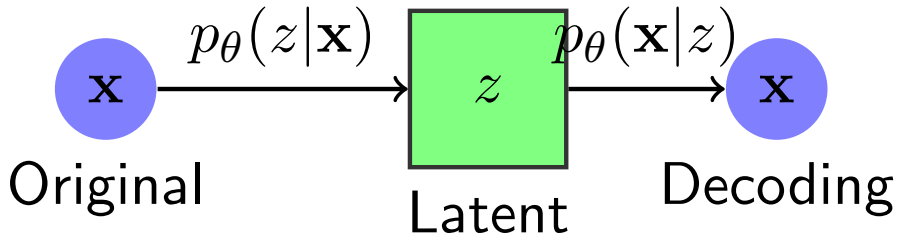
- training data lies nearby a low-dimensional manifold.
- a corrupted example is obtained by applying a perturbation of original example
- The model should learn to *project them back* to the manifold

Modern Autoencoder

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

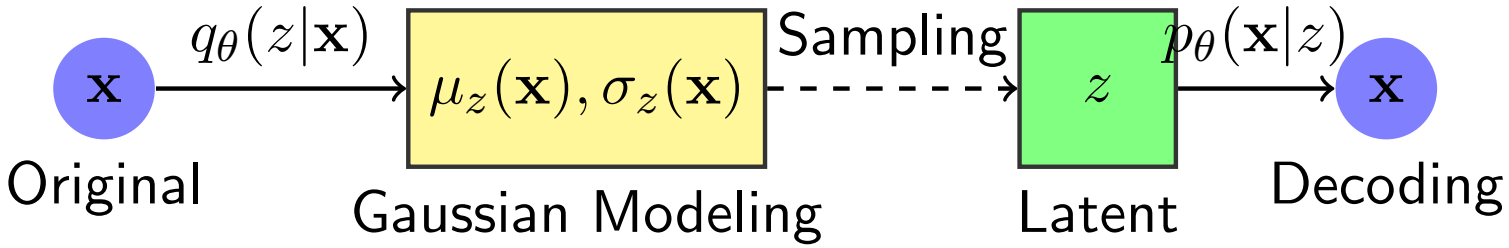
Contents

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 $q(z|\mathbf{x}) = \mathcal{N}(z; \mu_z(x), \sigma_z(x))$

Variational Autoencoder



- Gaussian modeling (diagonal covariance matrix to avoid problems in high dimensionality)
- Training via maximum likelihood of $p(\mathbf{x})$
- Learning the parameters θs via backpropagation?

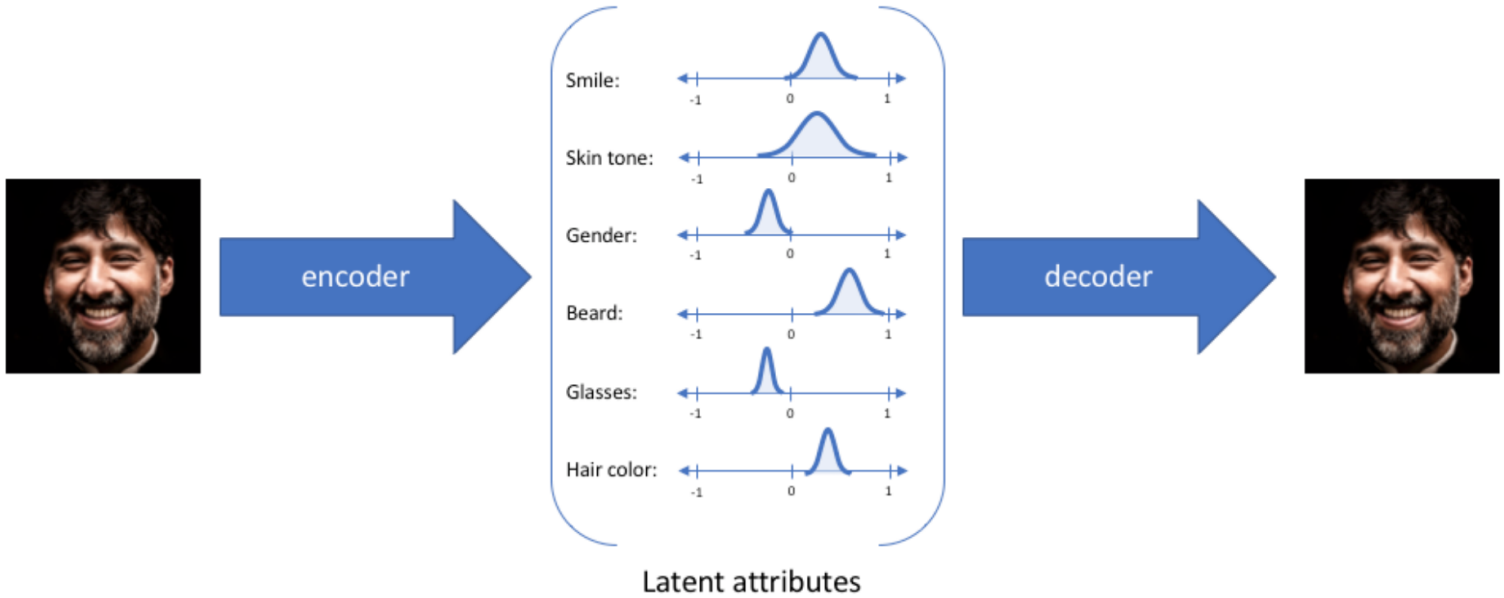


Figure: Latent Space Intuition (Variational Case). Source: `https://www.jeremyjordan.me/variational-autoencoders/`

Training via maximum likelihood

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

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

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

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

Reparametrization trick

Backpropagation is not possible through random sampling!

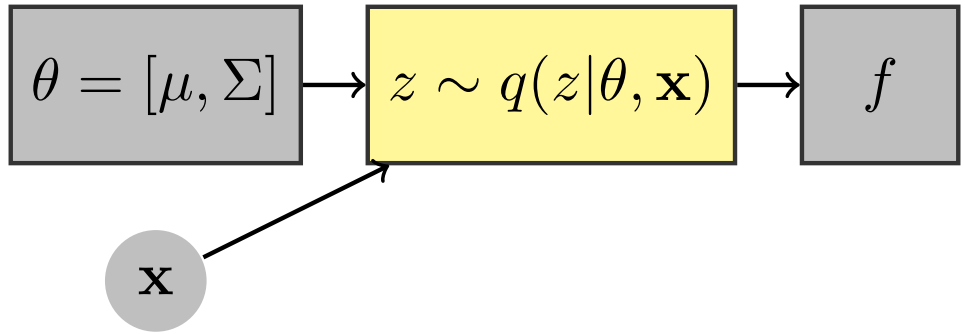


Figure: Original Formulation

Reparametrization trick

Backpropagation is not possible through random sampling!

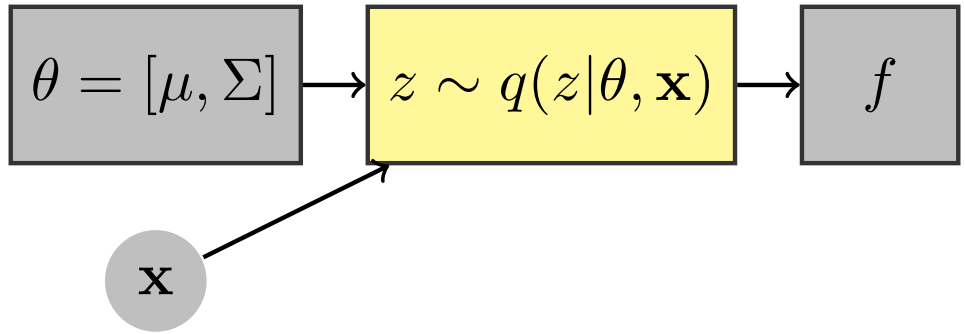


Figure: Original Formulation

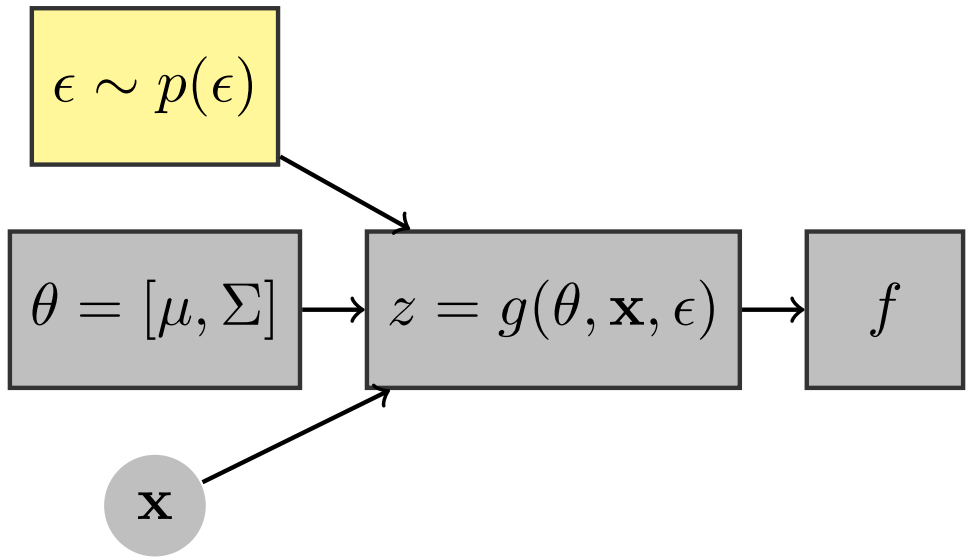


Figure: Reparametrization trick. Backpropagation

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

Dimensionality Reduction by AE

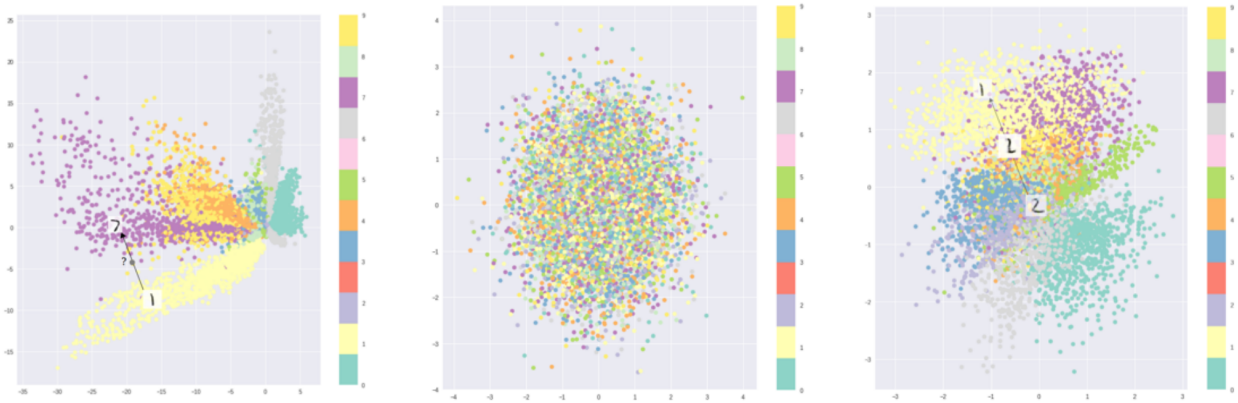
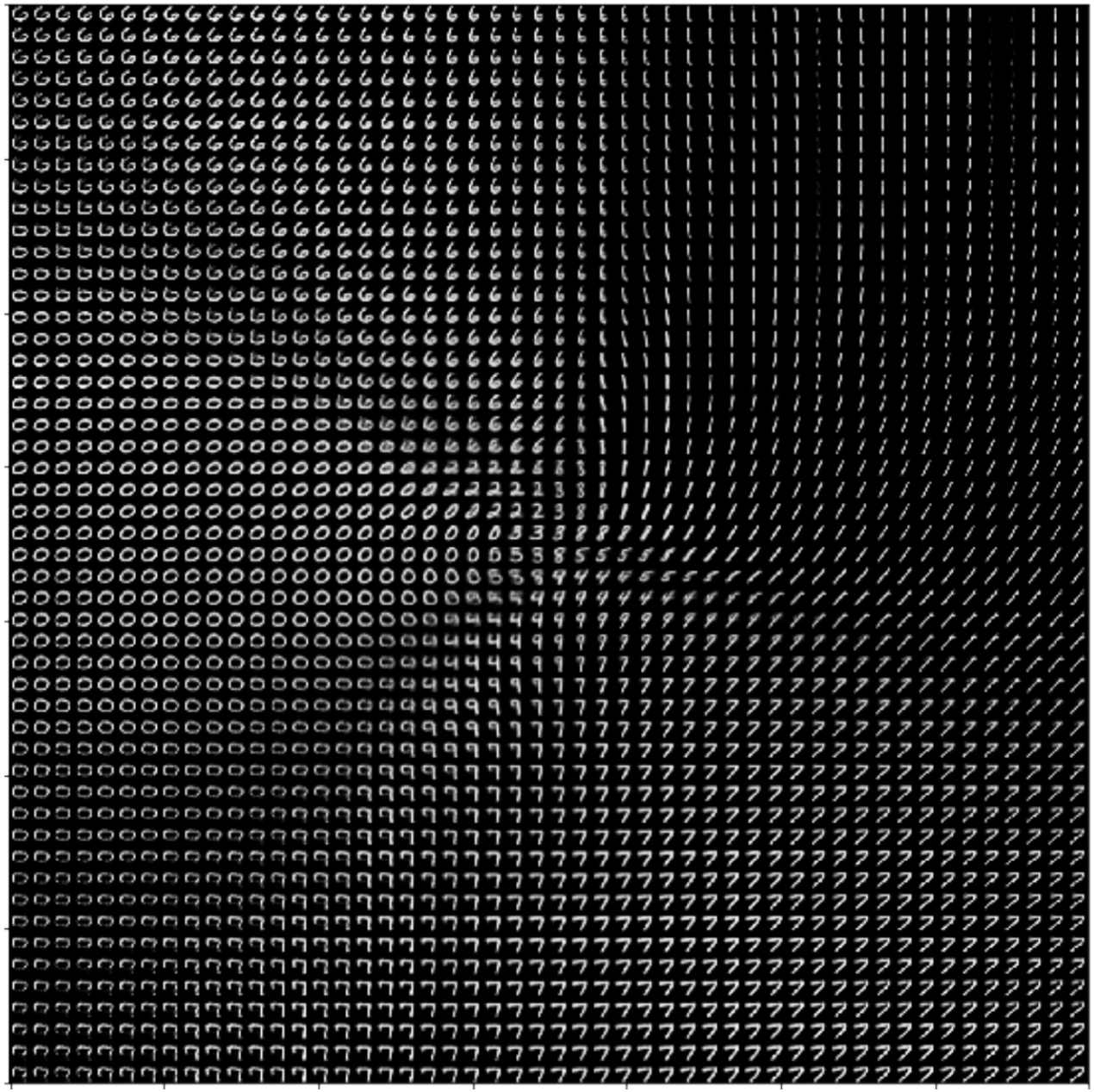


Figure: **Left**: AE, **Center**: Only KL in VAC, **Right**: VAE

Source:
`https://towardsdatascience.com/intuitively-understanding-va`

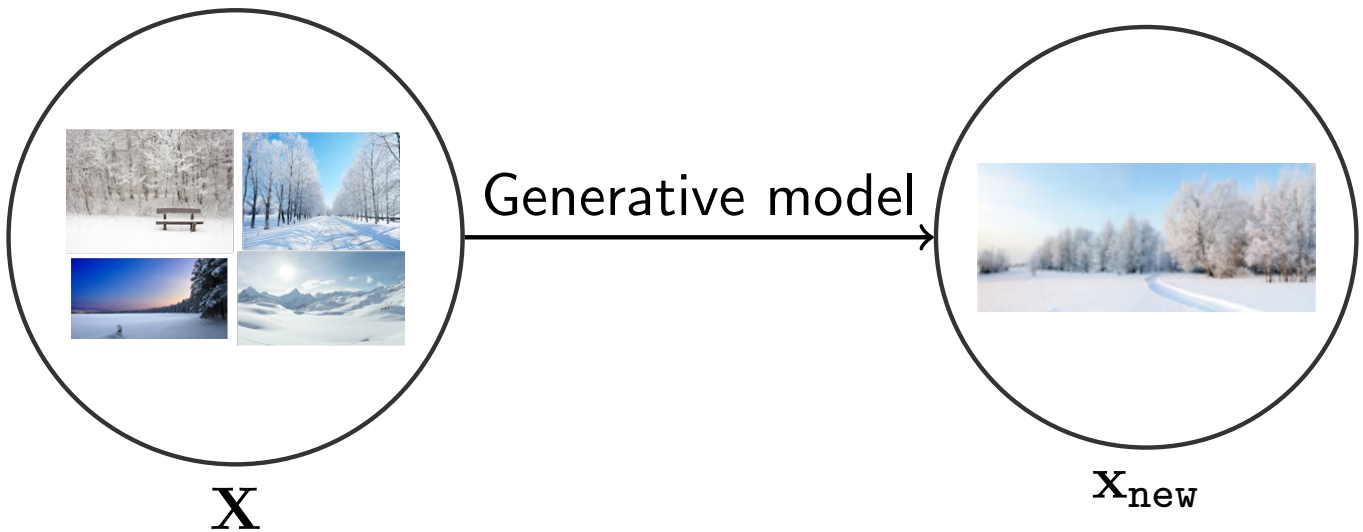
Example Latent Space for MNIST



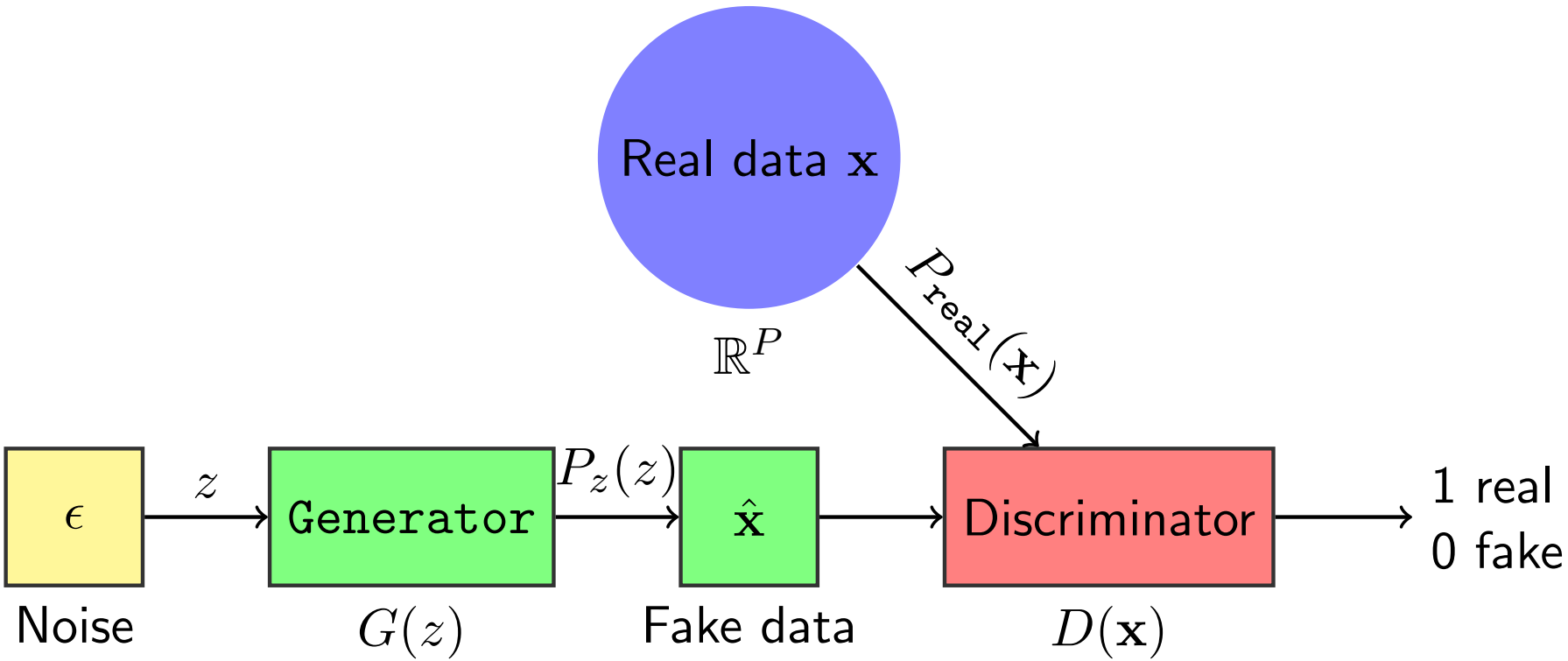
Contents

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?



Generative Adversarial Networks (GANs) [Goodfellow et al., 2014]



Generative Adversarial Networks (GANs)

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

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

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:

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

Generative Adversarial Networks (GANs)

Questions: Optimal Discriminator? Optimal Generator? We define:

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

What should be an optimal discriminator?

Generative Adversarial Networks (GANs)

Questions: Optimal Discriminator? Optimal Generator? We define:

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

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?

Generative Adversarial Networks (GANs)

Questions: Optimal Discriminator? Optimal Generator? We define:

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

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?

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

This is called the *minimax formulation*, since the generator and discriminator are playing a *zero-sum game* against each other:

$$\min_G \max_D V(D, G) \tag{4}$$

Generative Adversarial Networks (GANs)

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

An optimal generator G^* should give $P_{g(\mathbf{x})} = P_{\texttt{real}}(\mathbf{x})$.
Now, given a generator G, we can compute the optimal discriminator by:

$$
\begin{aligned}
\frac{\partial V(G,D)(\mathbf{x})}{\partial D(\mathbf{x})} &= P_{\texttt{real}}(\mathbf{x})\frac{1}{D(\mathbf{x})} + P_g(\mathbf{x})\frac{1}{1-D(\mathbf{x})} \\
&= \frac{P_{\texttt{real}}(\mathbf{x}) - (P_{\texttt{real}}(\mathbf{x}) + P_{\texttt{g}}(\mathbf{x}))D(\mathbf{x})}{D(\mathbf{x})(1-D(\mathbf{x}))}
\end{aligned}
$$

Setting $\frac{\partial V(G,D)(\mathbf{x})}{\partial D(\mathbf{x})} = 0$, we obtain $D^*(\mathbf{x}) = \frac{P_{\texttt{real}}(\mathbf{x})}{P_{\texttt{real}(\mathbf{x})+P_g(\mathbf{x})}}$

Global Optimal on Generative Adversarial Networks (GANs)

$$V(G,D) = \mathbb{E}_{\mathbf{x}\sim P_{\texttt{real}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{x\sim P_{g(\mathbf{x})}}[\log(1-D(\mathbf{x}))]$$
$$= \int_{\mathbf{x}} P_{\texttt{real}}(\mathbf{x}) \log D(\mathbf{x}) + P_{g(\mathbf{x})} \log(1-D(\mathbf{x}))d\mathbf{x}$$

We are interested in compute $V(G^*, D^*) =$

Global Optimal on Generative Adversarial Networks (GANs)

$$V(G,D) = \mathbb{E}_{\mathbf{x}\sim P_{\texttt{real}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{x\sim P_{g(\mathbf{x})}}[\log(1-D(\mathbf{x}))]$$
$$= \int_{\mathbf{x}} P_{\texttt{real}}(\mathbf{x}) \log D(\mathbf{x}) + P_{g(\mathbf{x})} \log(1-D(\mathbf{x}))d\mathbf{x}$$

We are interested in compute $V(G^*, D^*) = 2\log(1/2)$

Training GANs: Training the Discriminator:

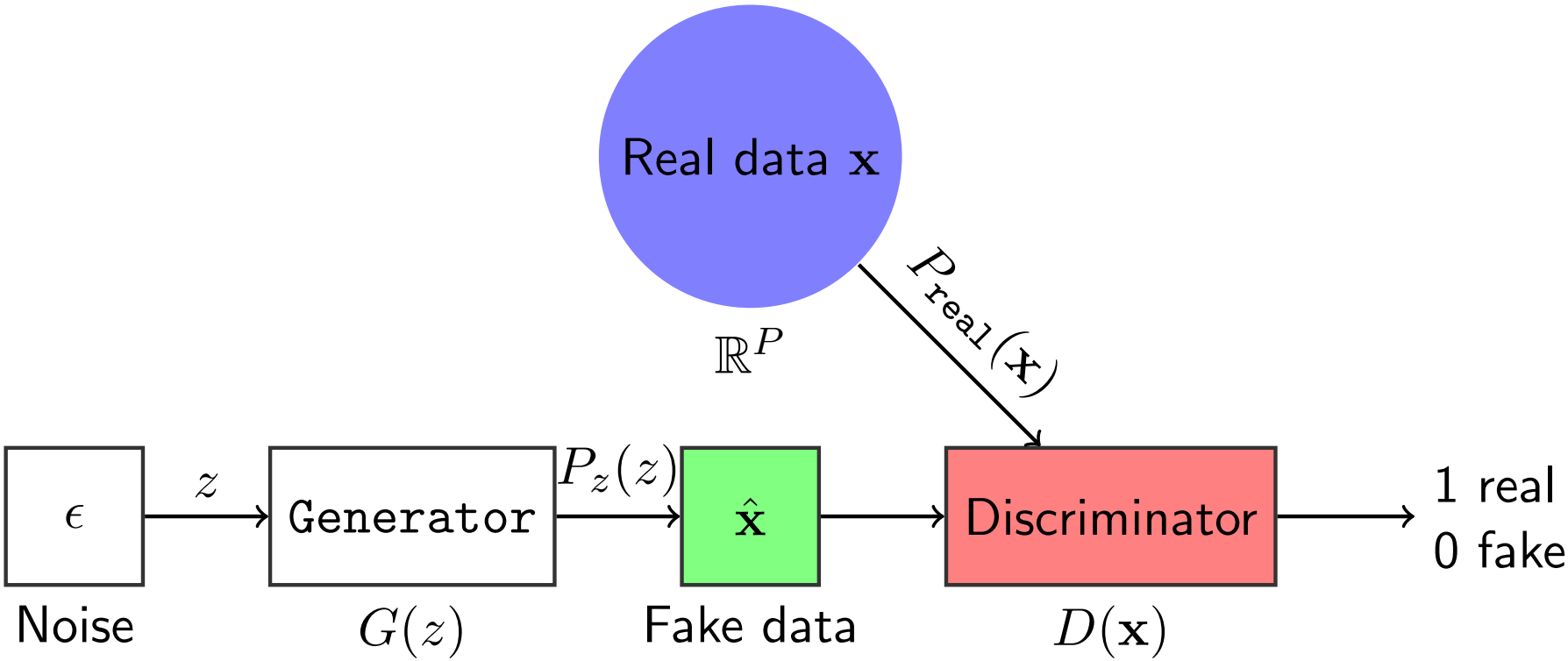


Figure: Only Discriminator should be updated

Training GANs: Training the Generator

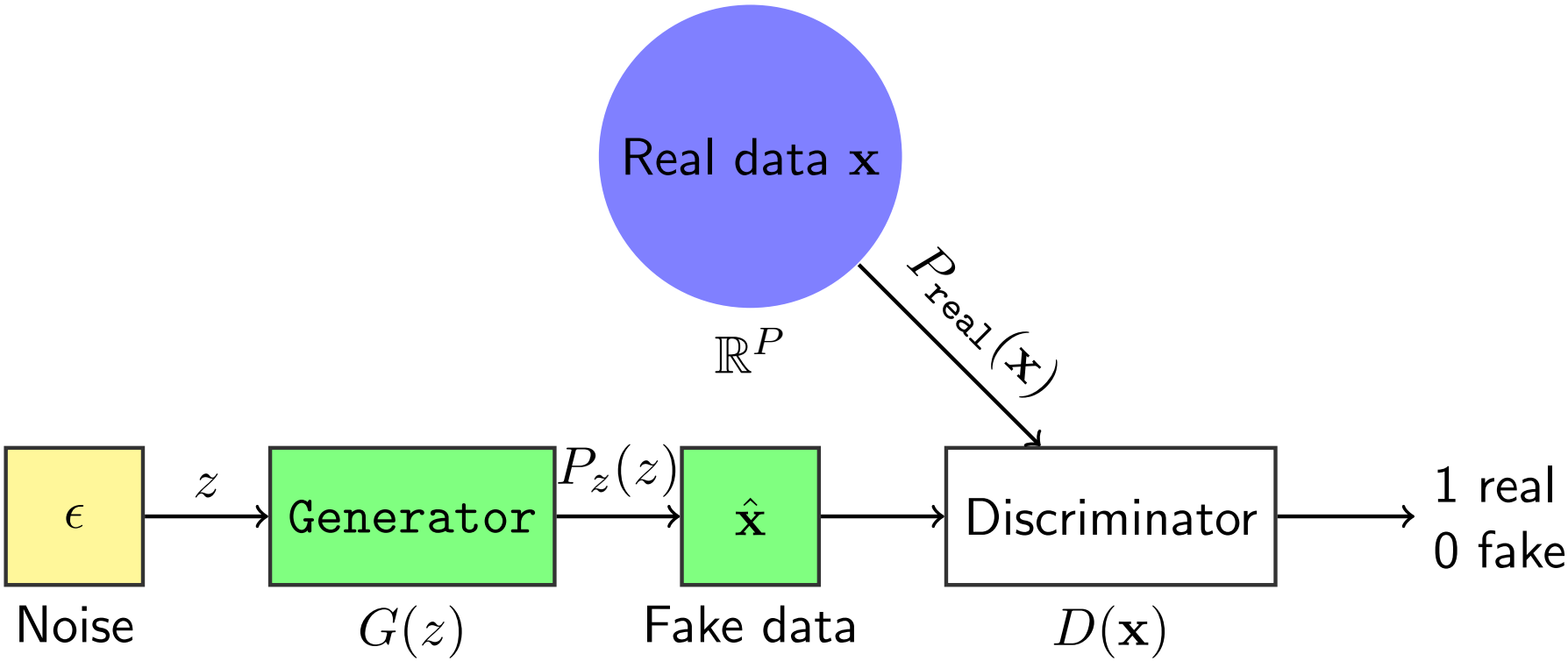


Figure: Discriminator should be set to "non-trainable" weights

How to Train a GAN?)

Bad news: GANs are hard to train. **Tips and tricks to make GANs work** `https://github.com/soumith/ganhacks`

- Use others loss functions.
- Avoid Sparse Gradients: ReLU, MaxPool
- Use Deep Convolutional Generative Adversarial Networks when you can. It works!
- Use SGD for discriminator and ADAM for generator
- Use Dropouts in G in both train and test phase
- ...

"In theory, there is no difference between theory and practice. In practice, there is."

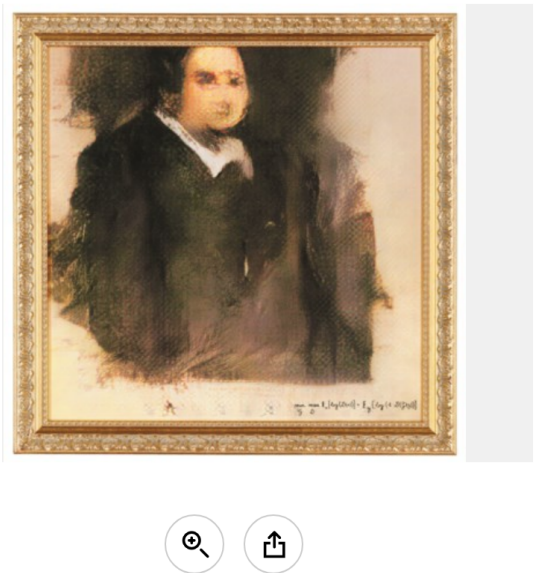
High-resolution GANs

Progressive Growing of GANs for Improved Quality, Stability, and Variation [Karras et al., 2017]
`https://www.youtube.com/watch?v=G06dEcZ-QTg`

Applications of GANs: Earn money!

Applications of GANs: Earn money!

23-25 Oct, 2018: Auction house "Christies" sold a portrait for 432,000 dollars that had been generated by a GANs



LOT 363

Edmond de Belamy, from La Famille de Belamy

Price realised
USD 432,500

Estimate
USD 7,000 - USD 10,000

Follow lot

+ Add to Interests

Edmond de Belamy, from *La Famille de Belamy*
generative Adversarial Network print, on canvas, 2018, signed with GAN model loss function in ink by the publisher, from a series of eleven unique images, published by Obvious Art, Paris, with original gilded wood frame
S. 27 ½ x 27 ½ in (700 x 700 mm.)

GANs to improve simulations

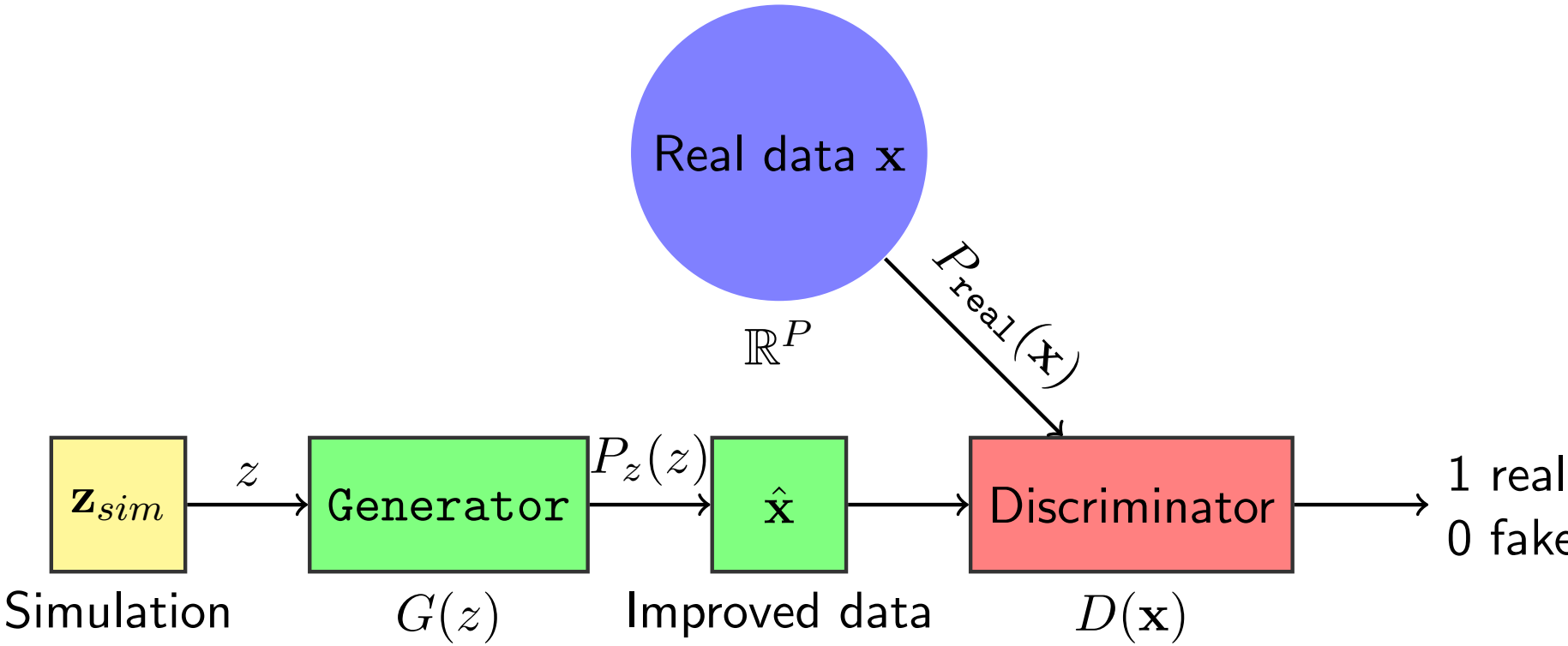


Figure: In this case, `Generator` is called `Refiner`

Application: Refine simulations

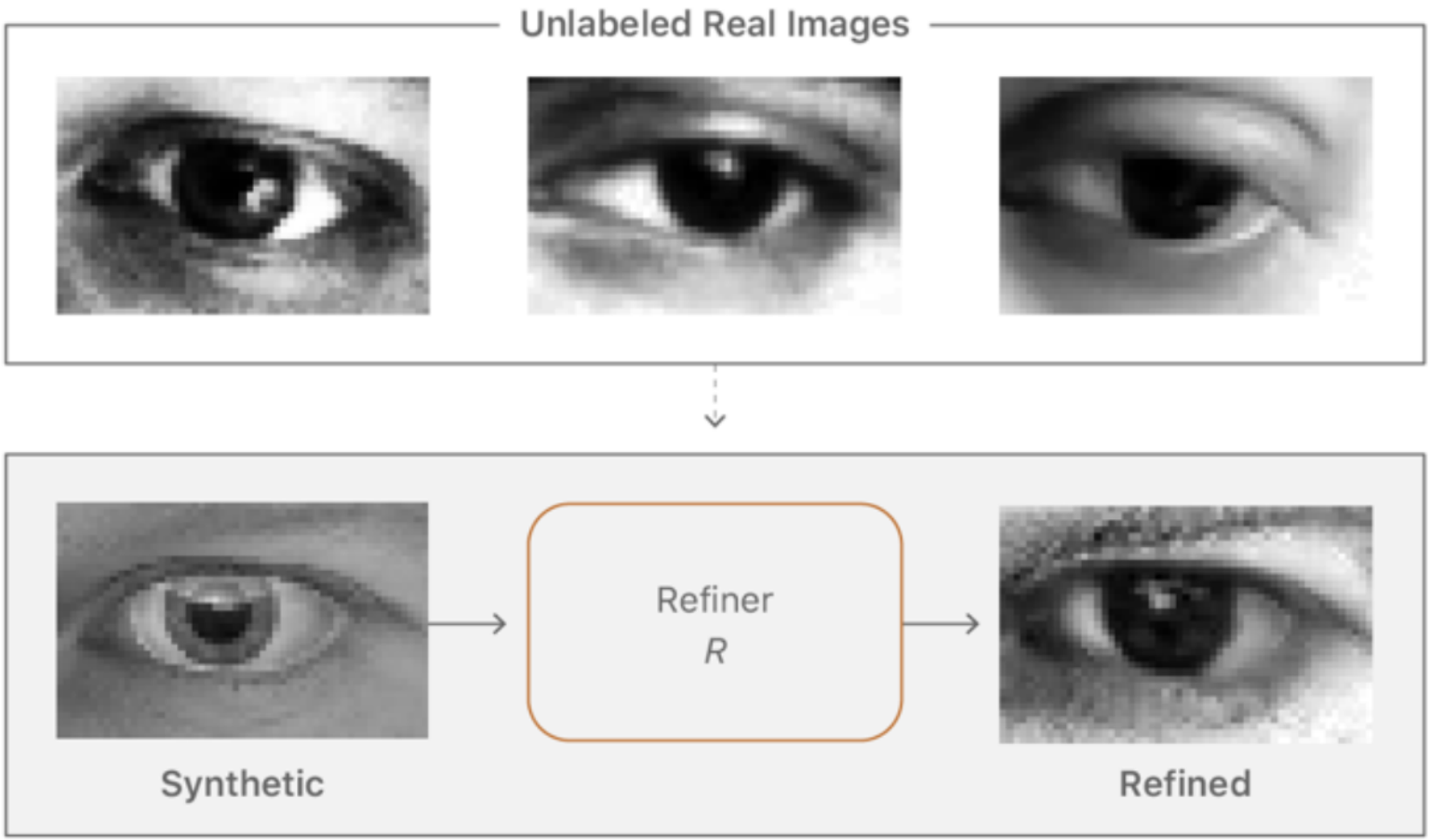
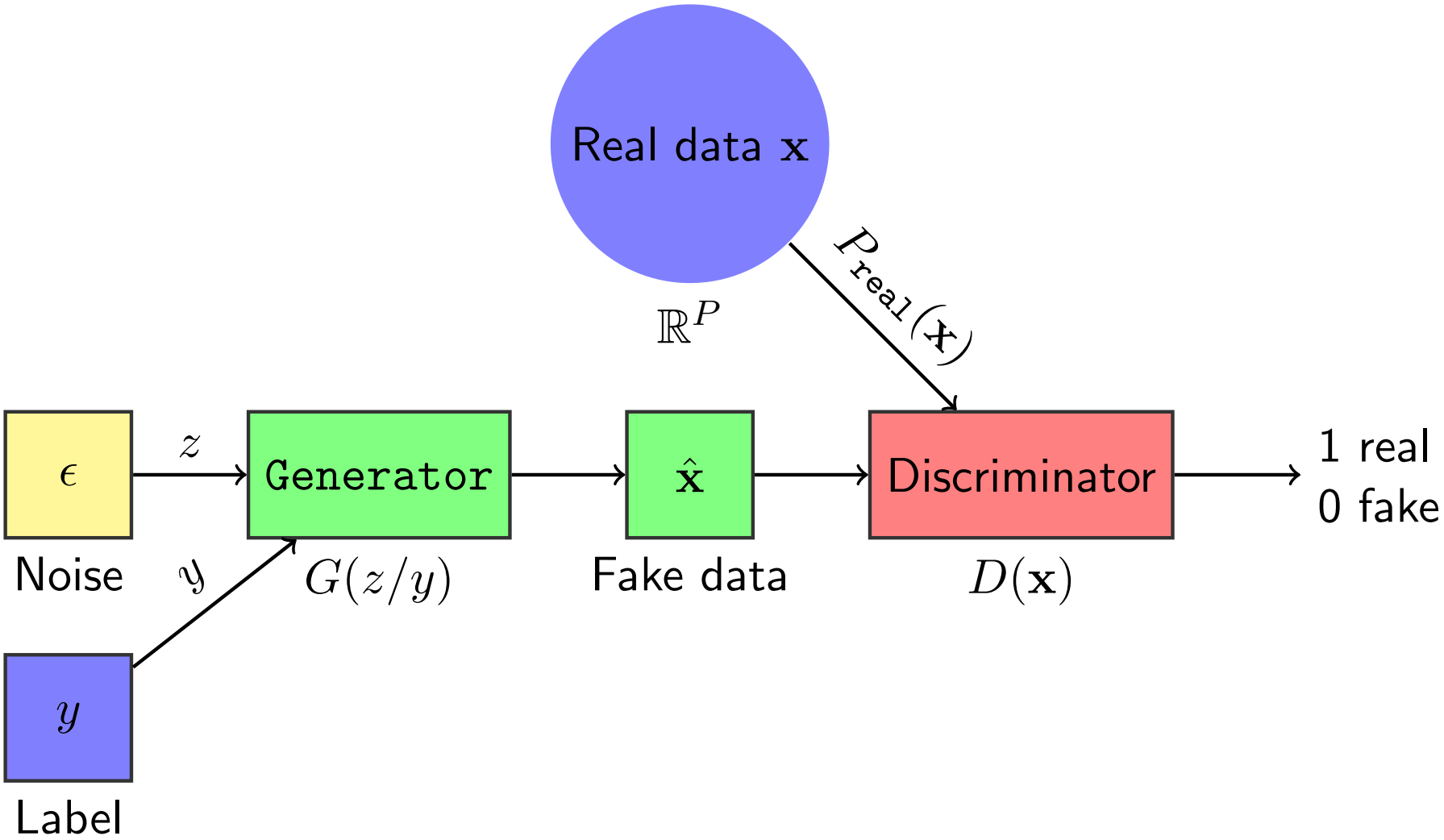


Figure: Learning from Simulated and Unsupervised Images through Adversarial Training [Shrivastava et al., 2017]

Conditional Generative Adversarial Networks (cGANs)



Example: cGANs

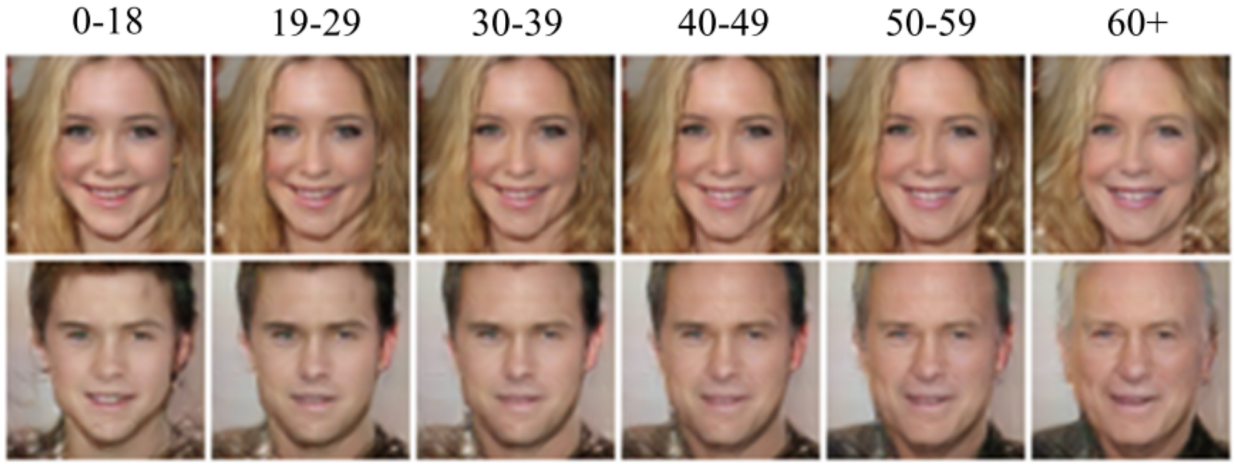


Figure: Face aging with conditional generative adversarial networks

[Antipov et al., 2017]

Pix2Pix transformation

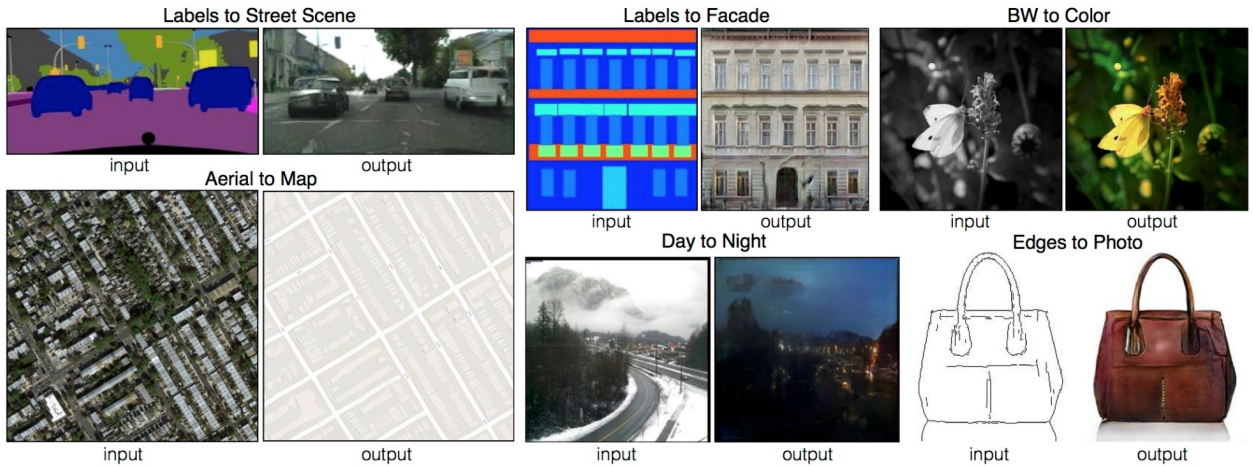


Figure: Example of Pix2Pix, a.k.a. image-to-image translation, [Isola et al., 2017]

GANs for Pix2Pix

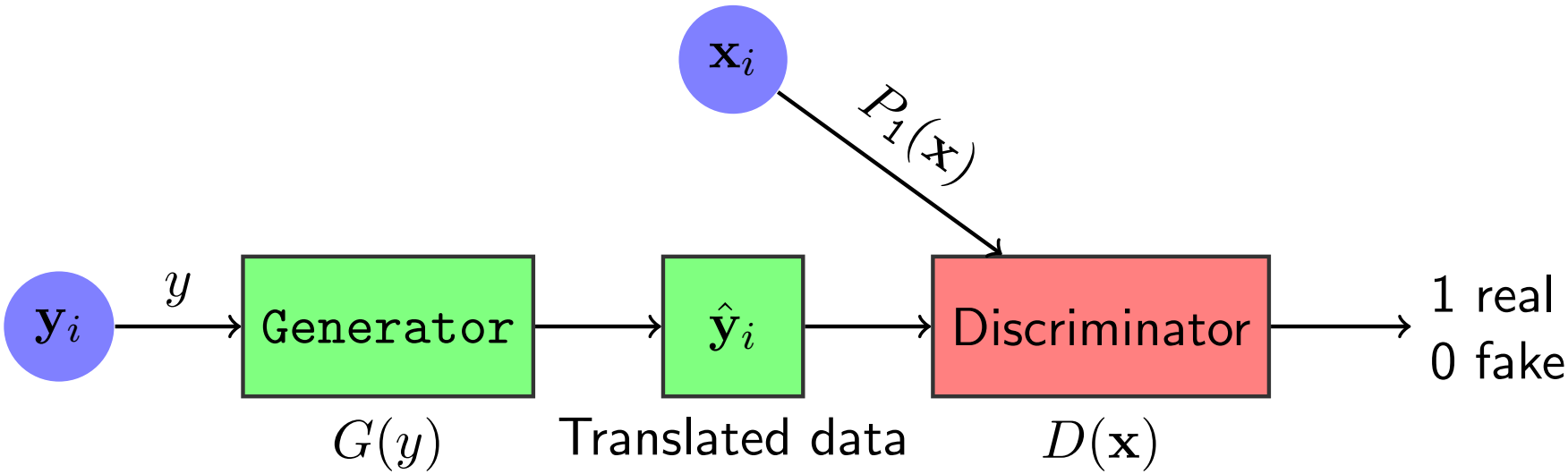


Figure: Training data are pairs of image: $(\mathbf{x}_i, \mathbf{y}_i)$ for $i \in 1, \ldots, N$. In this case, `Generator` is called `Translator`

Target Domain Transformation

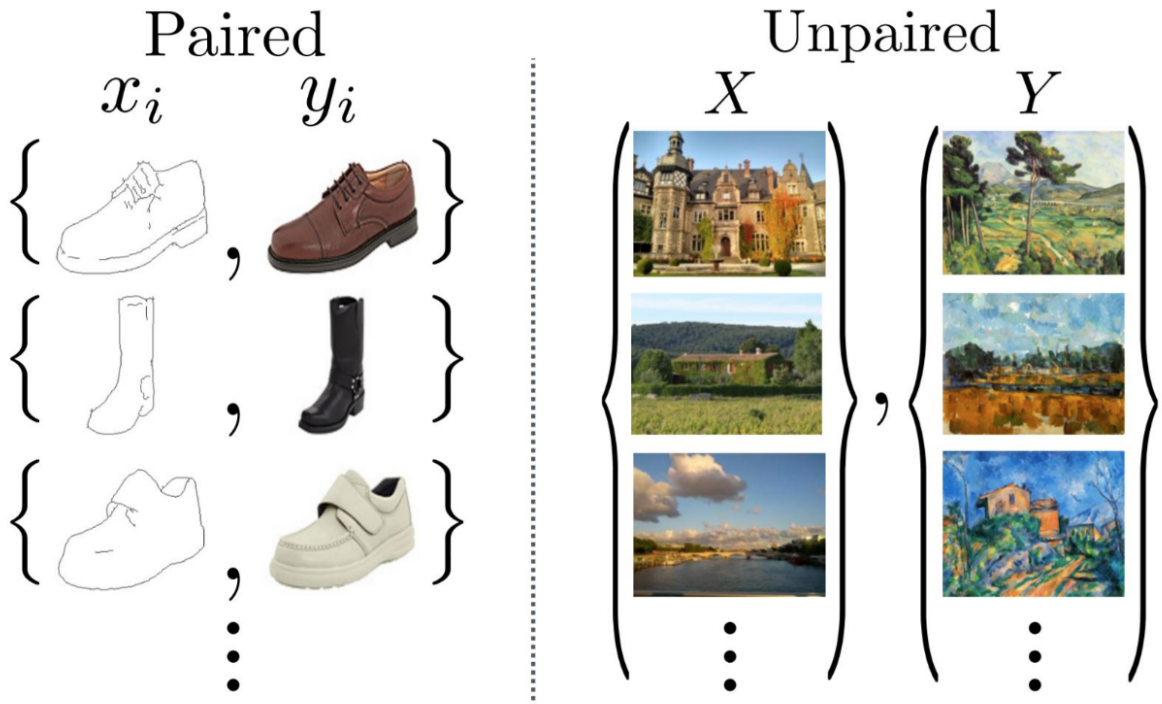


Figure: *Left*: Image to Image translation, *Right*: Source to Target domain transform

Cycle-Consistent Adversarial Networks

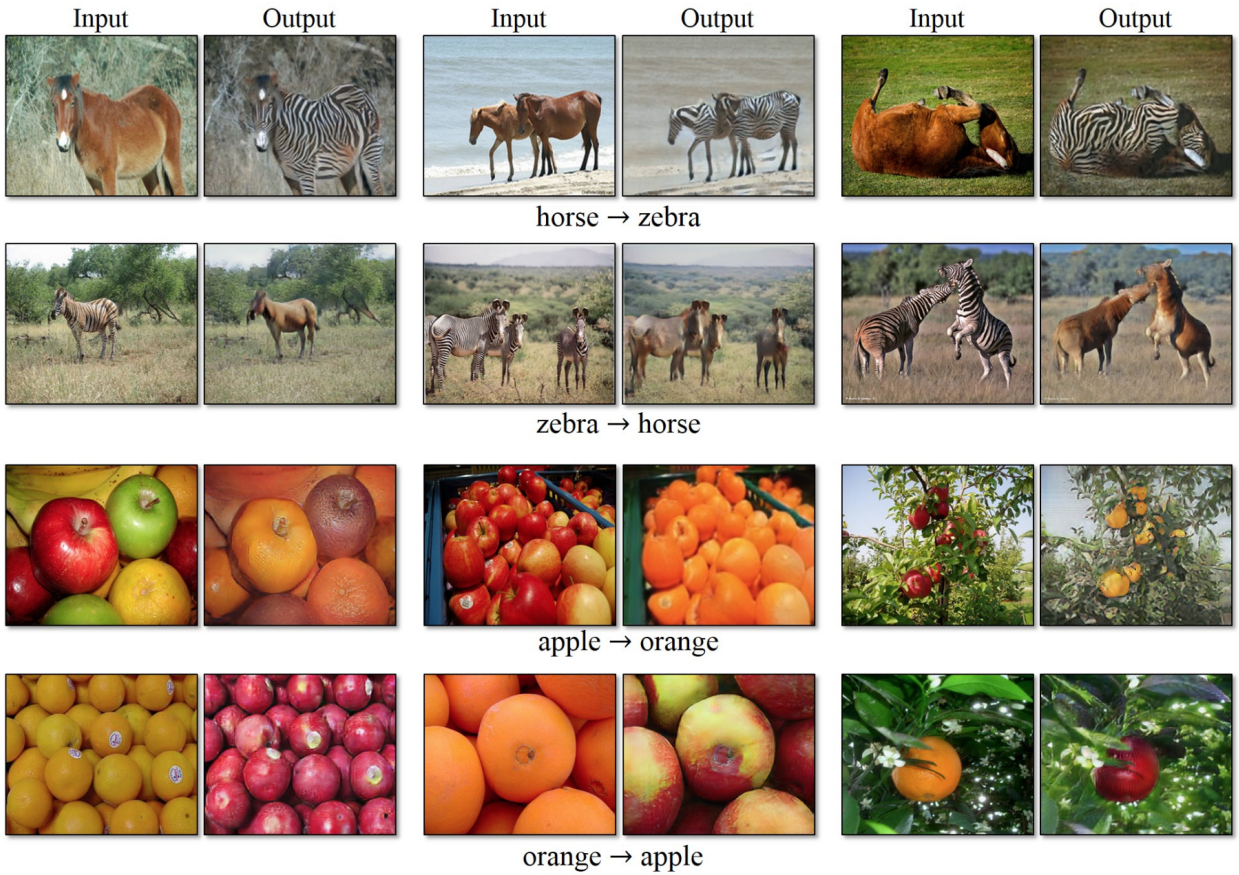


Figure: Unpaired image-to-image translation [Zhu et al., 2017]

Cycle-Consistent Adversarial Networks

We have to train two GANs for Pix2Pix transform:

- First: GANs to translate data from domain one $\mathbf{X}_1$ to domain two represented by $\mathbf{X}_2$, denoted by $V(G_1, D_1, \mathbf{X}_1, \mathbf{X}_2)$
- Second: GANs to translate data from domain one $\mathbf{X}_2$ to domain two represented by $\mathbf{X}_1$, denoted by $V(G_2, D_2, \mathbf{X}_2, \mathbf{X}_1)$

Cycle-Consistent Adversarial Networks

We have to train two GANs for Pix2Pix transform:

- First: GANs to translate data from domain one $\mathbf{X}_1$ to domain two represented by $\mathbf{X}_2$, denoted by $V(G_1, D_1, \mathbf{X}_1, \mathbf{X}_2)$
- Second: GANs to translate data from domain one $\mathbf{X}_2$ to domain two represented by $\mathbf{X}_1$, denoted by $V(G_2, D_2, \mathbf{X}_2, \mathbf{X}_1)$
- Include: a Cycle consistency loss V_{cyc} to force that the composition of two `Generators` to reconstruct input image, i.e $G_1(G_2(\mathbf{X}_2)) \approx \mathbf{X}_2$ and $G_2(G_1(\mathbf{X}_1)) \approx \mathbf{X}_1$

In general the full objective is:

$$L(G_1, G_2, D_1, D_2) = V(G_1, D_1, \mathbf{X}_1, \mathbf{X}_2) + V(G_2, D_2, \mathbf{X}_2, \mathbf{X}_1) + \lambda V_{cyc}(G_1, G_2)$$

Contents

Adversarial Examples [Goodfellow et al., 2015]

- 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 $||\mathbf{x} - \mathbf{x}^{adv}|| \leq \delta$.
- An human user would still visually consider the adversarial input $\mathbf{x}^{adv}$ similar to or the same as the benign input x
- Usually, we are interested in adversarial for benign sample $\mathbf{x}$, i.e., sample were the model gives a correct predictions.

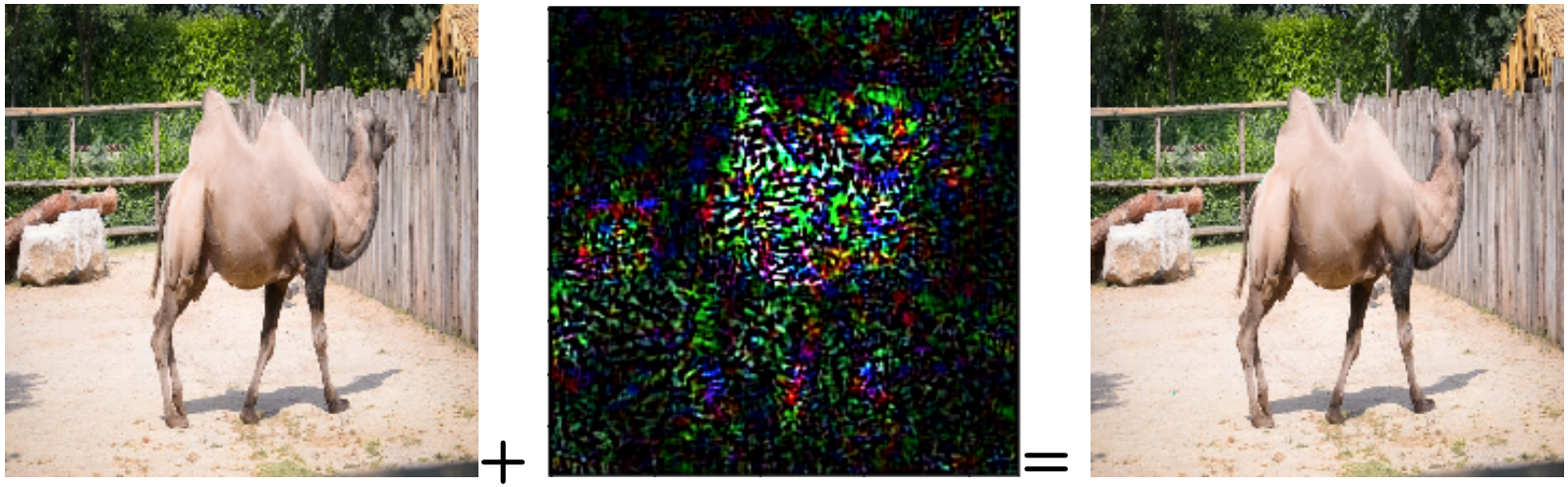


Figure: $\mathbf{x} + \epsilon = \mathbf{x}^{adv}$

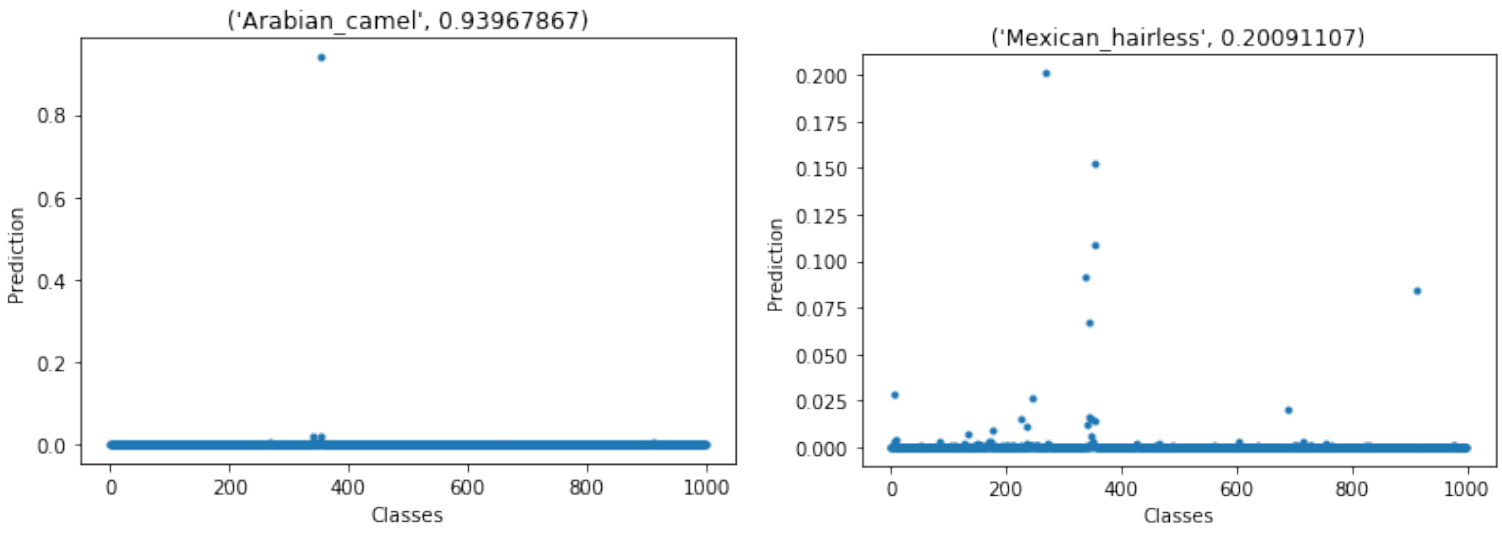


Figure: VGG19 Prediction for: $\mathbf{x}$ and $\mathbf{x}^{adv}$

Gradient based Methods

- Fast Gradient Sign Method [Goodfellow et al., 2015] Their perturbation can be expressed as

$$\eta = \epsilon \cdot sign(\nabla_{\mathbf{x}} L(\mathbf{x}, y_{\texttt{true}})) \tag{5}$$

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

- Iterative Gradient Sign Method [Kurakin et al., 2016] $\mathbf{x}_0^{adv} = \mathbf{x}$, iteratively repeat $\mathbf{x}_{t+1}^{adv} = \texttt{CLIP}(\mathbf{x}_t^{adv} + \epsilon \cdot sign(\nabla_{\mathbf{x}_t^{adv}} L(\mathbf{x}_t^{adv}, y_{\texttt{true}})))$

Gradient based Methods

- Fast Gradient Sign Method [Goodfellow et al., 2015] Their perturbation can be expressed as

$$\eta = \epsilon \cdot sign(\nabla_{\mathbf{x}} L(\mathbf{x}, y_{\texttt{true}})) \tag{5}$$

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

- Iterative Gradient Sign Method [Kurakin et al., 2016] $\mathbf{x}_0^{adv} = \mathbf{x}$, iteratively repeat $\mathbf{x}_{t+1}^{adv} = \texttt{CLIP}(\mathbf{x}_t^{adv} + \epsilon \cdot sign(\nabla_{\mathbf{x}_t^{adv}} L(\mathbf{x}_t^{adv}, y_{\texttt{true}})))$

To avoid overfitting! and obtain more robust model.... Adversarial Training = Data Augmentation with adversarial examples

Universal Perturbation

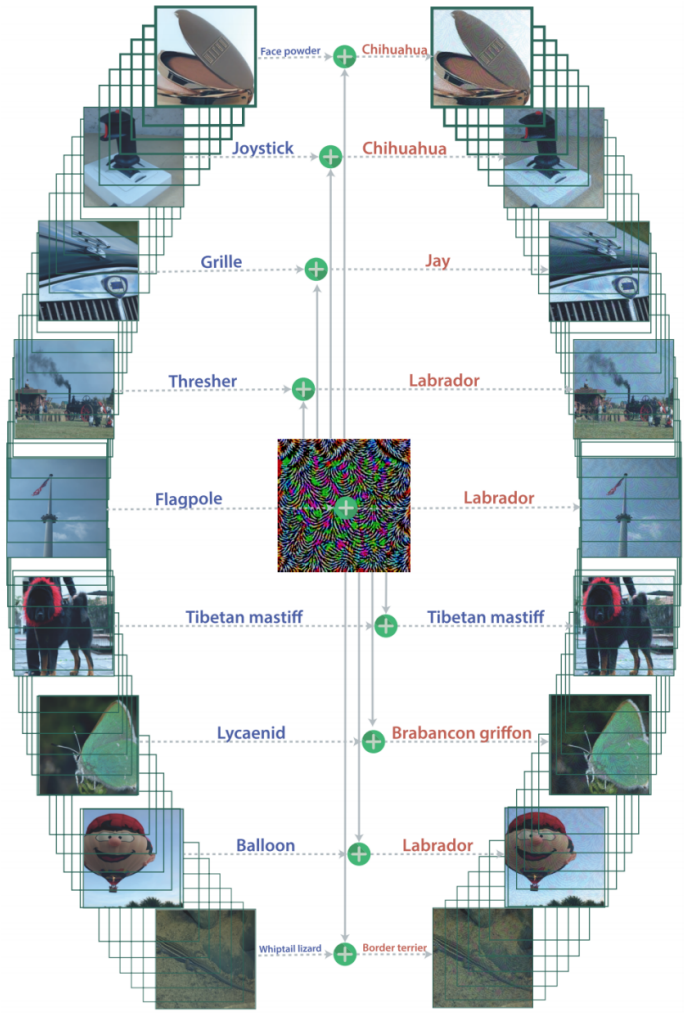


Figure: Universal adversarial perturbations [Moosavi-Dezfooli et al., 2017]

White box Attacks

'Egyptian_cat', 0.3396838

'lynx', 0.47152225

'jay', 0.96423554

VGG19

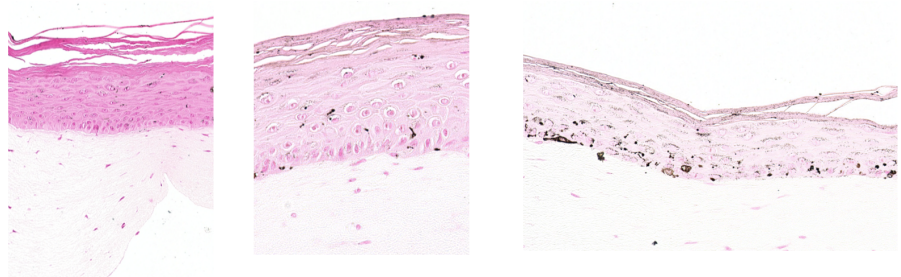
'plastic_bag', 0.54238236

'electric_ray', 0.8287997

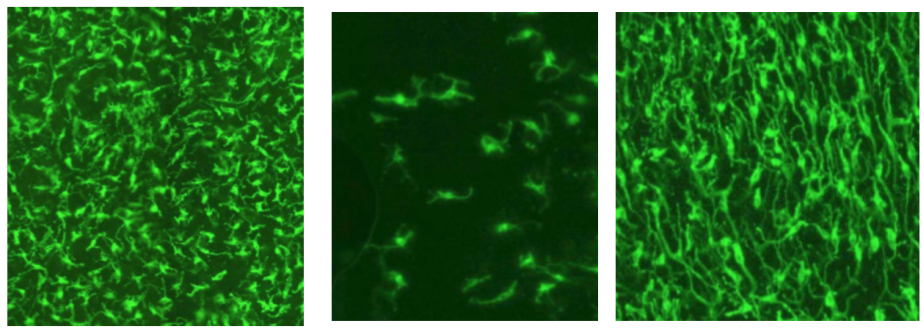
Figure: White Box Attacks [Akhtar and Mian, 2018]

Job Offers: Master Internships

1. Cycle-Consistent Adversarial Networks for Biological Images (Mines + L'Oreal)

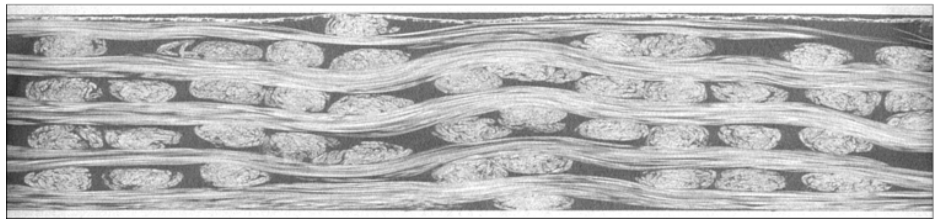


2. Learning to Count by Deep Learning for Biological Images (Mines + LOreal)



Job Offers: Master Internships

1. Deep Detection with temporal information on Material Science (Mines + Safran)



2. Unsupervised Deep Learning Model for Manifold Data (PostDoc), Mines+Thales)

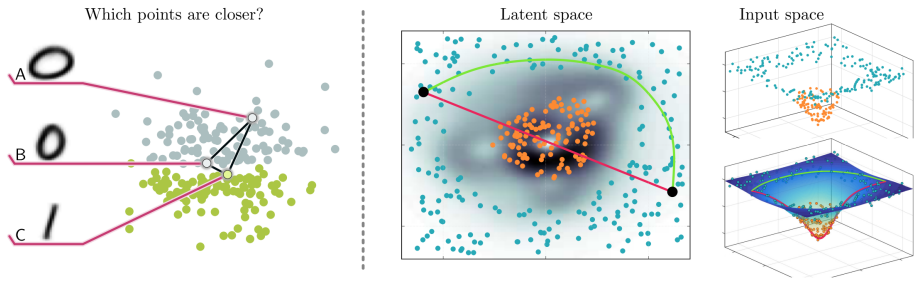


Figure: Image from: `Latent Space Oddity: On the Curvature of Deep Generative Models`

Contents

References I

[Akhtar and Mian, 2018] Akhtar, N. and Mian, A. (2018). Threat of adversarial attacks on deep learning in computer vision: A survey. *https://arxiv.org/pdf/1801.00553.pdf*.

[Antipov et al., 2017] Antipov, G., Baccouche, M., and Dugelay, J.-L. (2017). Face aging with conditional generative adversarial networks. In *IEEE International Conference on Image Processing (ICIP)*, pages 2089–2093. IEEE.

[Bengio et al., 2007] Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160.

References II

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[Goodfellow et al., 2015] Goodfellow, I., Shlens, J., and Szegedy, C. (2015). Explaining and Harnessing Adversarial Examples.

[Isola et al., 2017] Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[Karras et al., 2017] Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2017). Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*.

References III

[Kurakin et al., 2016] Kurakin, A., Goodfellow, I., and Bengio, S. (2016). Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*.

[Moosavi-Dezfooli et al., 2017] Moosavi-Dezfooli, S.-M., Fawzi, A., Fawzi, O., and Frossard, P. (2017). Universal adversarial perturbations. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533.

[Shrivastava et al., 2017] Shrivastava, A., Pfister, T., Tuzel, O., Susskind, J., Wang, W., and Webb, R. (2017). Learning from simulated and unsupervised images through adversarial training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

References IV

[Vincent et al., 2008] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM.

[Vincent et al., 2010] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408.

[Zhu et al., 2017] Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.