Introduction to Similarity Learning, Adversarial Examples, Autoencoders and GANs

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

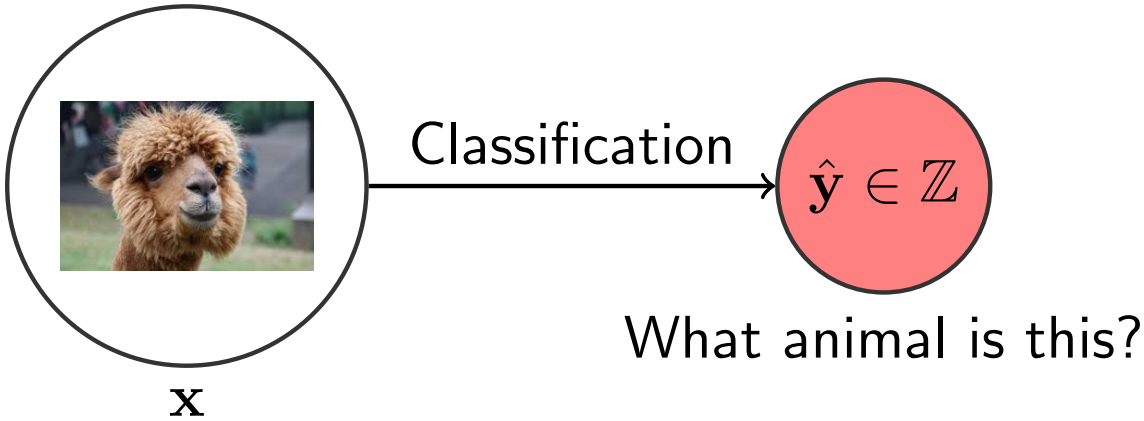
MINES Paris
PSL Research University
Center for Mathematical Morphology
February 10, 2023

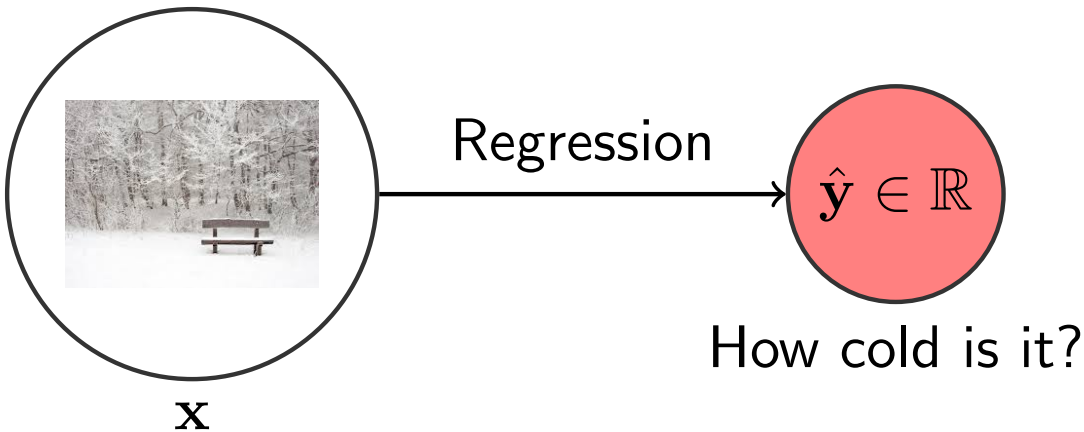
Contents

Contents

Supervised Learning

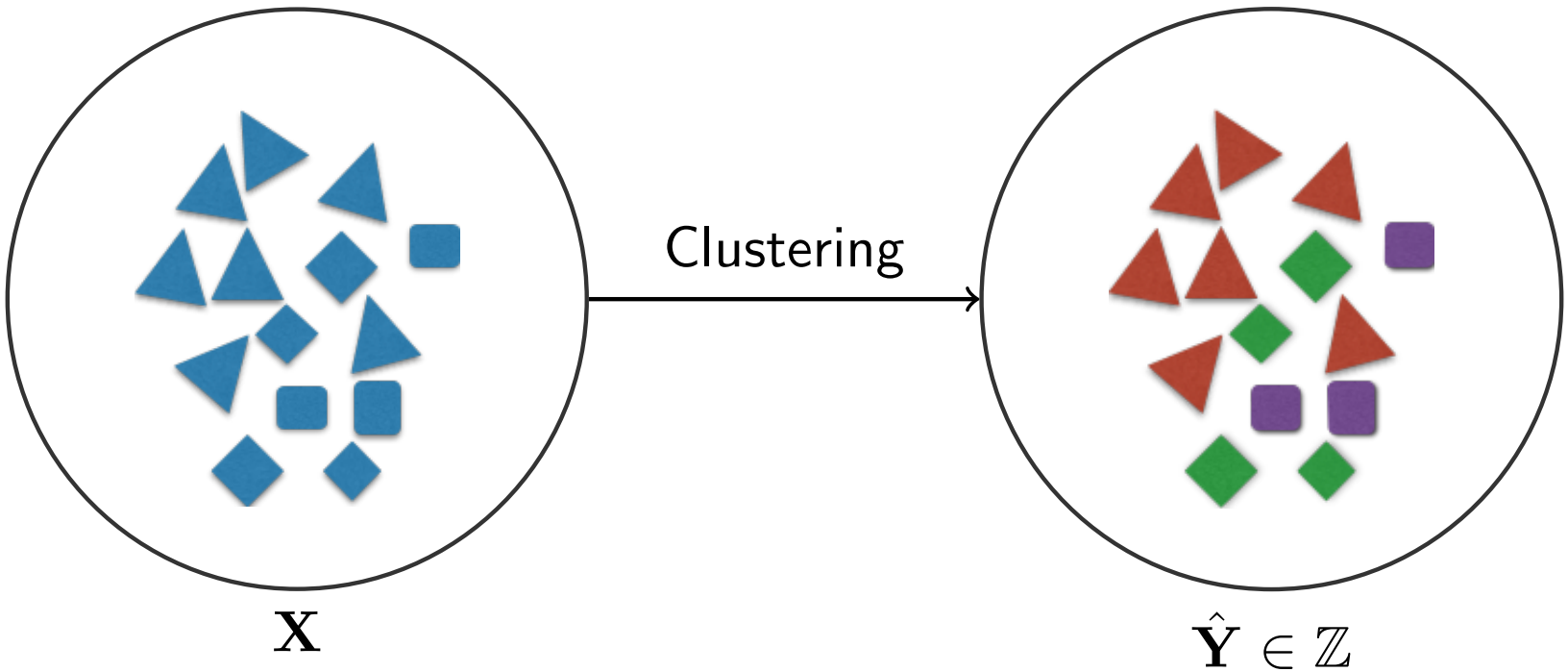
Given a labeled dataset $(\mathbf{X}, \mathbf{Y})$, we would like to learn a mapping from data space to label space.





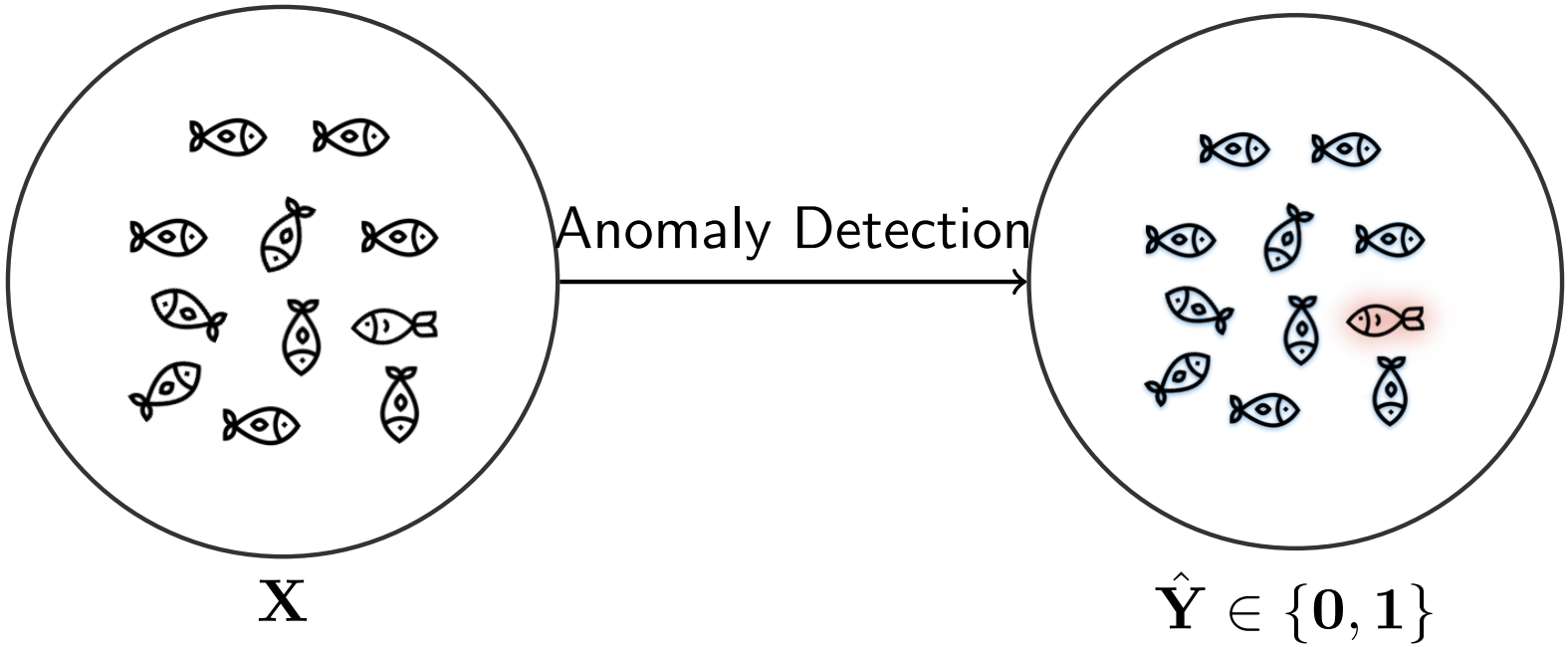
Unsupervised Learning: Clustering

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



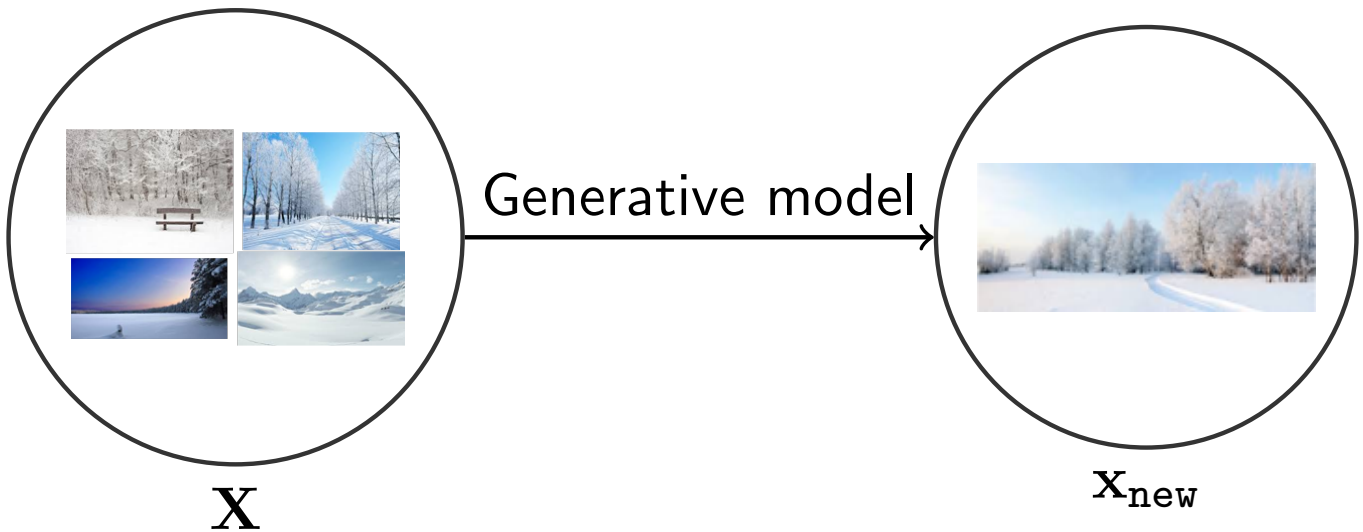
Anomaly detection

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



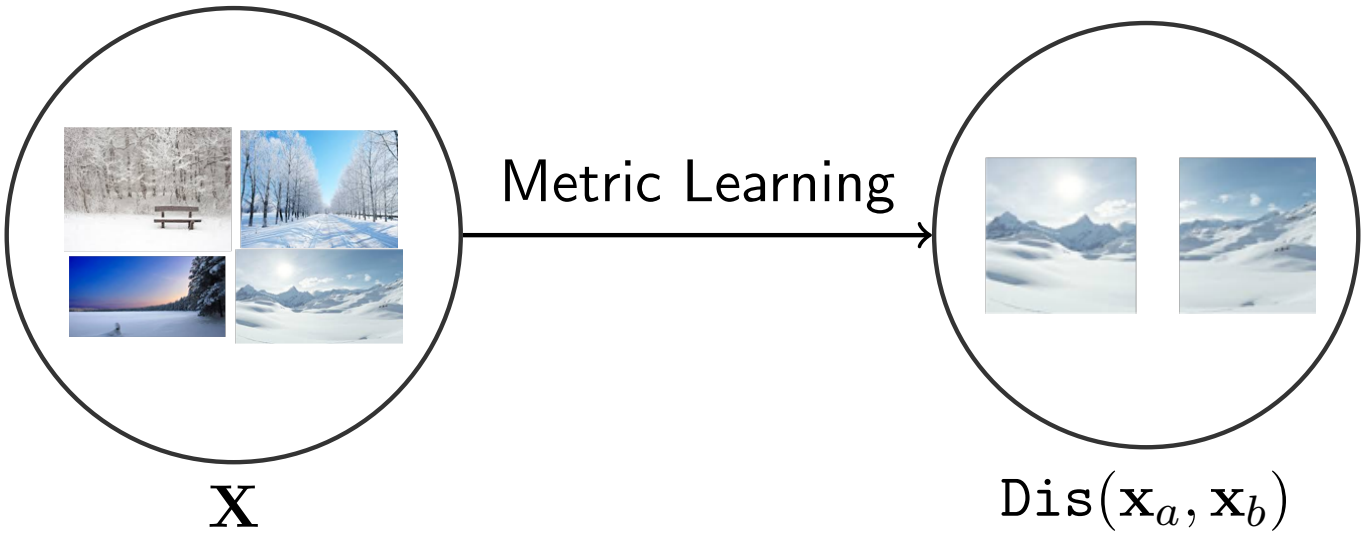
Unsupervised learning: Generative Models

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



Unsupervised learning: Metric Learning

Given an unlabeled dataset $(\mathbf{X})$, we would like to learn: How to estimate the distance between two objects of $\mathbf{X}$?



1. $\mathtt{Dis}(\mathbf{x}, \mathbf{x}') = 0 \iff \mathbf{x} = \mathbf{x}'$ (Identity)
2. $\mathtt{Dis}(\mathbf{x}, \mathbf{x}) = 0, \forall \mathbf{x}$ (Non-negative)
3. $\mathtt{Dis}(\mathbf{x}, \mathbf{x}') = \mathtt{Dis}(\mathbf{x}', \mathbf{x}), \forall \mathbf{x}, \mathbf{x}$ (Symmetry)
4. $\mathtt{Dis}(\mathbf{x}, \mathbf{x}'') > \mathtt{Dis}(\mathbf{x}, \mathbf{x}' + \mathtt{Dis}(\mathbf{x}', \mathbf{x}'')$ (Triangle Inequality)

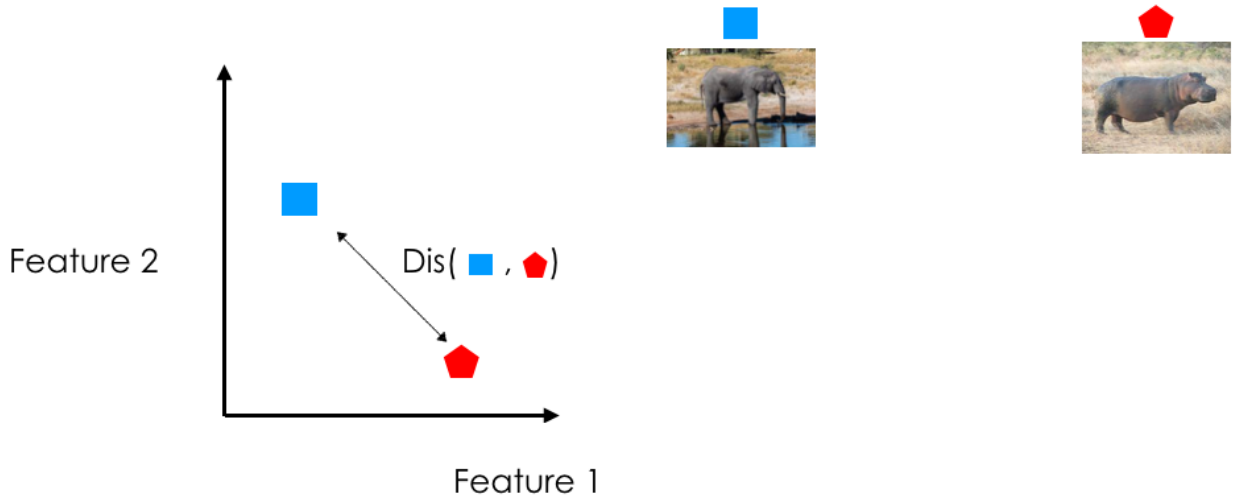
DL for Metric Learning

Metric Learning has been used in many cases, for example:

1. Retrieval of Objects: Given a database, find the k-closest point a given example.
2. Clustering.
3. Anomaly detection.
4. Pretraining Networks.

Contents

Introduction to Metric Learning

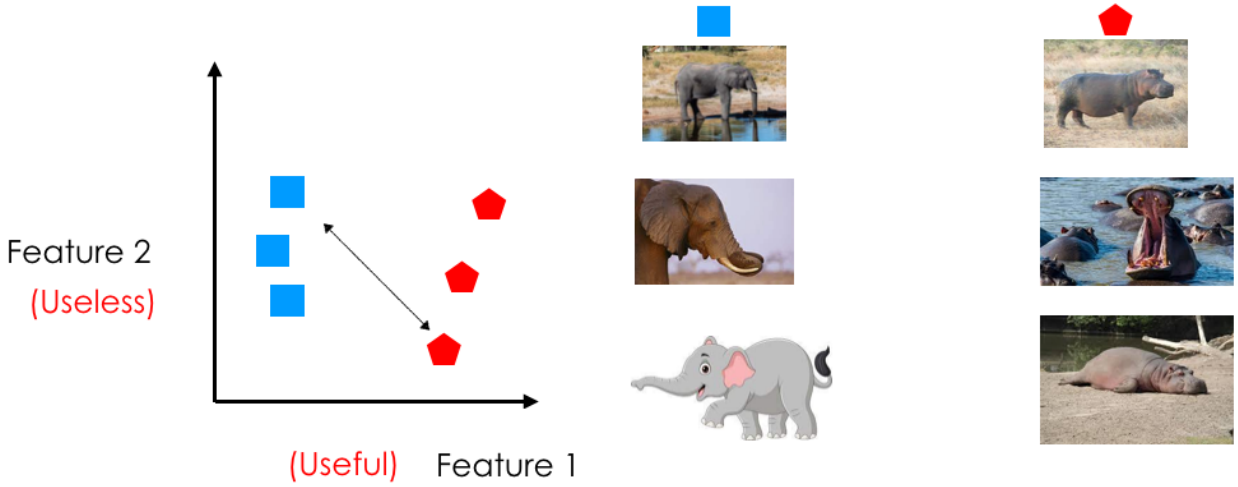


The natural idea is to compute the Euclidean distance:

$$\texttt{Dis}(\mathbf{x},\mathbf{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots} = \sqrt{(\mathbf{x}-\mathbf{y})^T(\mathbf{x}-\mathbf{y})}$$

Introduction to Metric Learning

How to compare observations?



$$
\begin{aligned}
\texttt{Dis}(\mathbf{x},\mathbf{y}) &= \sqrt{w_1(x_1-y_1)^2+w_2(x_2-y_2)^2+\ldots} \\
&= \sqrt{(\mathbf{L}(\mathbf{x}-\mathbf{y}))^T(\mathbf{L}(\mathbf{x}-\mathbf{y}))} \\
&= \sqrt{(\mathbf{x}-\mathbf{y})^T\mathbf{L}^T\mathbf{L}(\mathbf{x}-\mathbf{y})} \\
&:= \sqrt{(\mathbf{x}-\mathbf{y})^T\mathbf{M}(\mathbf{x}-\mathbf{y})}
\end{aligned}
$$

Introduction to Metric Learning

$$
\begin{aligned}
\texttt{Dis}(\mathbf{x},\mathbf{y}) &= \sqrt{w_1(x_1-y_1)^2+w_2(x_2-y_2)^2+\ldots} \\
&= \sqrt{(\mathbf{L}(\mathbf{x}-\mathbf{y}))^T(\mathbf{L}(\mathbf{x}-\mathbf{y}))} \\
&= \sqrt{(\mathbf{x}-\mathbf{y})^T\mathbf{L}^T\mathbf{L}(\mathbf{x}-\mathbf{y})} \\
&:= \sqrt{(\mathbf{x}-\mathbf{y})^T\mathbf{M}(\mathbf{x}-\mathbf{y})}
\end{aligned}
$$

The goal in metric learning is to learn $\mathbf{M}$. Using the fact that a PSD[1] matrix $\mathbf{M}$ can always be decomposed as $\mathbf{M}=\mathbf{L}^T\mathbf{L}$ for some $\mathbf{L}$, one can show that both parameterizations are equivalent. In practice, an algorithm may thus solve the metric learning problem with respect to either $\mathbf{M}$ or $\mathbf{L}$

[1] Positive Semidefinite, i.e., $x^TMx \geq 0 \forall x$

Siamese Neural Network

Given a pair of similar examples $(\mathbf{x}_i, \mathbf{x}_j)$, a Siamese neural network (sometimes called a *twin neural network*) is an artificial neural network that uses the same weights while working in tandem on two different input vectors to compute comparable output vectors.

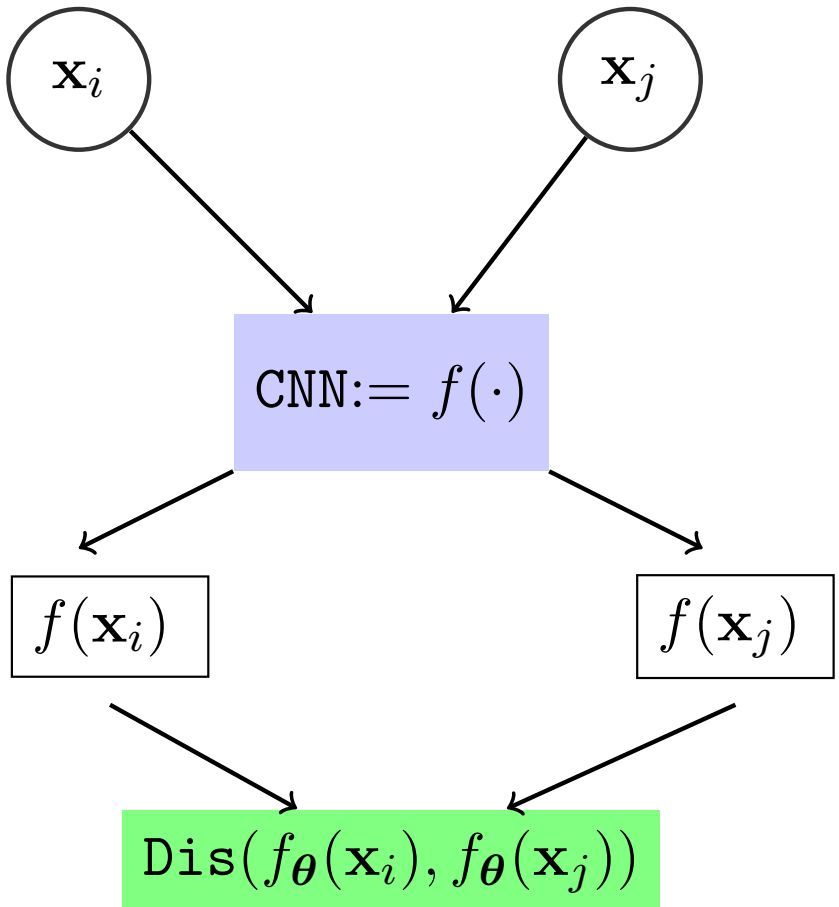


Figure: Scheme of Siamese Network

How to define the distance?

- A model is used to predict the position in feature space.
- A common way is to use a norm of the difference between the encoding of these two images, i.e.,
  $\texttt{Dis}(\mathbf{x}_i, \mathbf{x}_j) = ||f_{\boldsymbol{\theta}}(\mathbf{x}_i) - f_{\boldsymbol{\theta}}(\mathbf{x}_j)||_2^2$
- The important question is how to train the model?
  1. Loss functions
  2. how to define the training protocol.

Contents

How do we train a Siamese neural network?

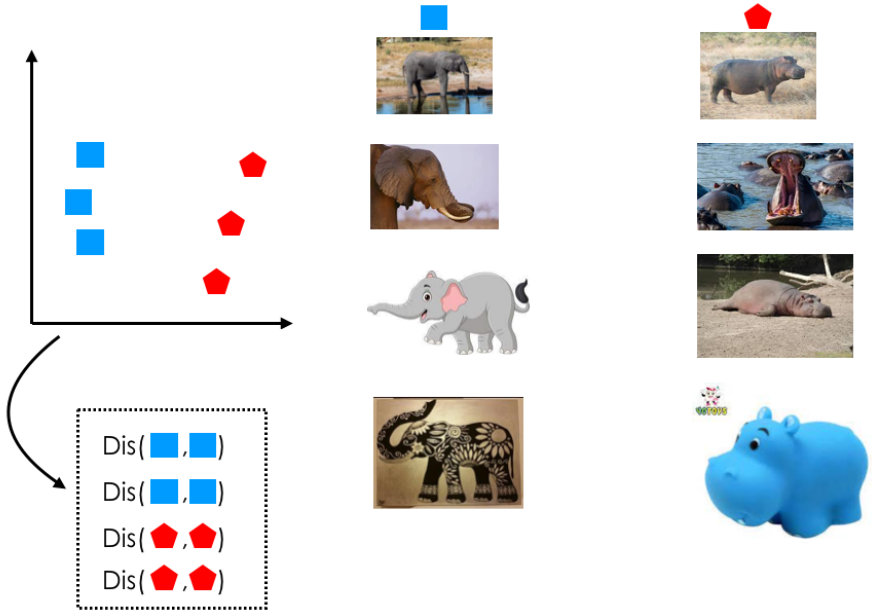


Figure: Note that the CNN, has parameters $\boldsymbol{\theta}$. A naive approach is to minimize the loss function $||f_{\boldsymbol{\theta}}(\mathbf{x}_1) - f_{\boldsymbol{\theta}}(\mathbf{x}_2)||_2^2$ for all the positive pairs

How do we train a Siamese neural network?

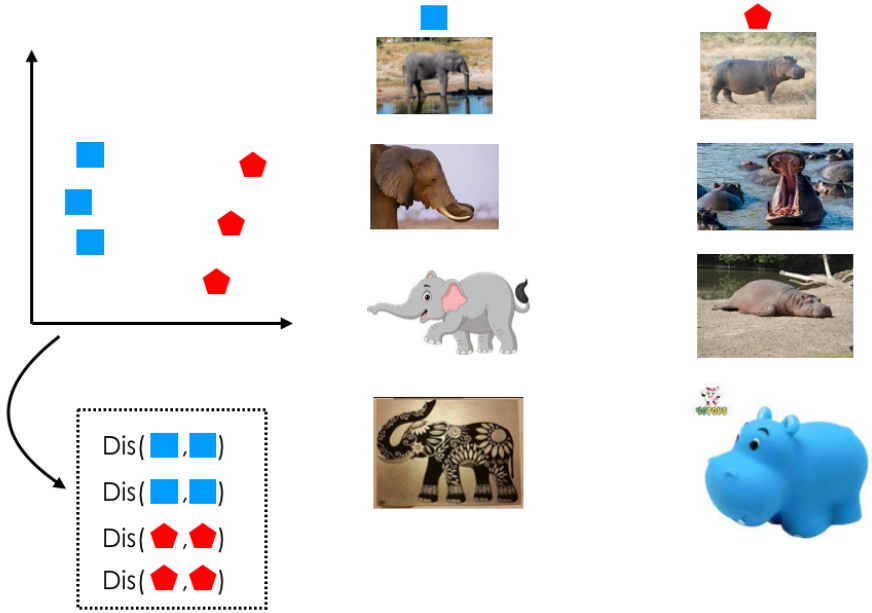


Figure: Note that the CNN, has parameters $\boldsymbol{\theta}$. A naive approach is to minimize the loss function $||f_{\boldsymbol{\theta}}(\mathbf{x}_1) - f_{\boldsymbol{\theta}}(\mathbf{x}_2)||_2^2$ for all the positive pairs

- The model could learn to *embed every input to the same point*, i.e. predict a constant as output. $(\boldsymbol{\theta} = 0)$
- "Neural collapsing". A possible solution is to use a regularization of the projection space to avoid the prediction as a constant.

Triplet Loss

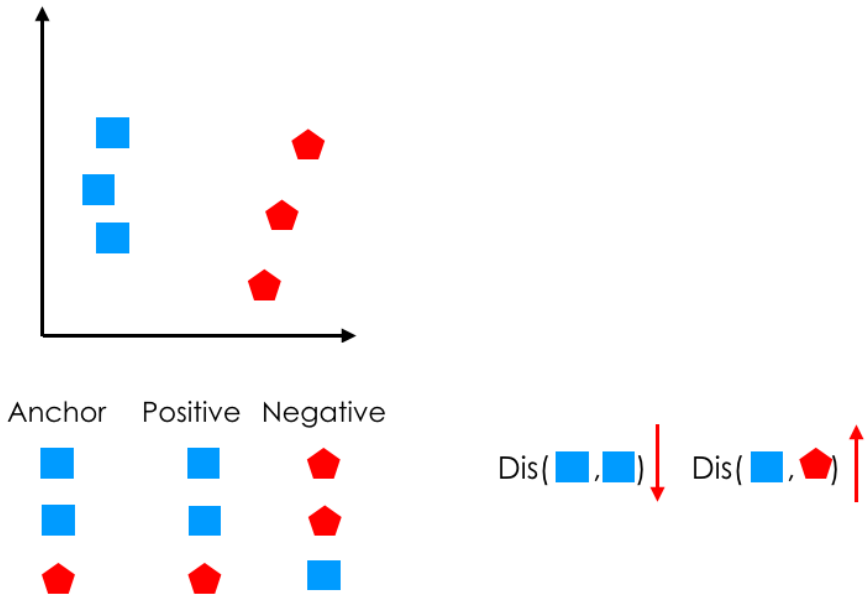


Figure: Triplet Loss: $L(a_i, p_i, n_i) = \max(\texttt{Dis}_{ap} - \texttt{Dis}_{an} + \alpha, 0)$.

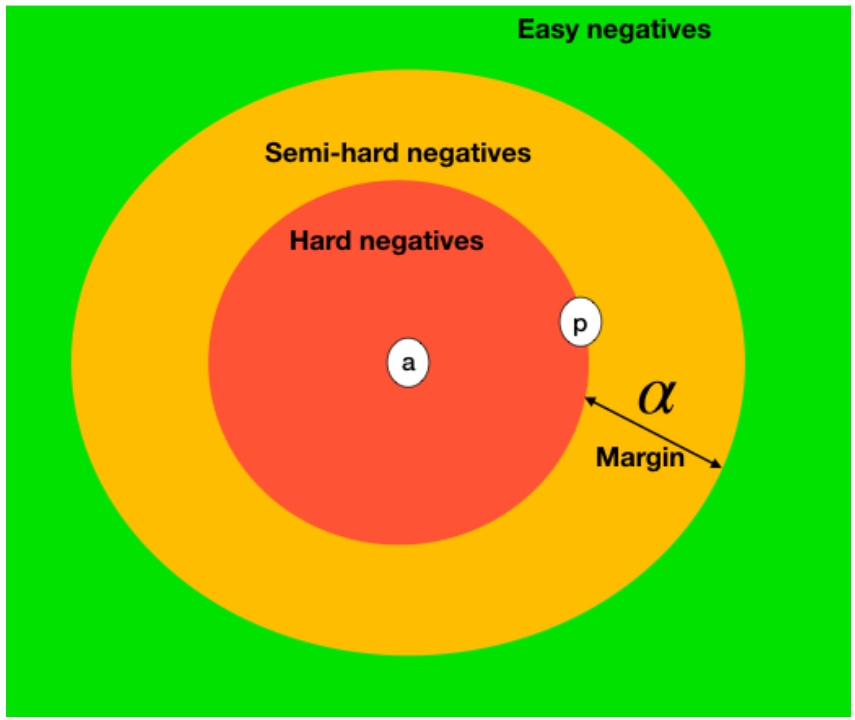
- As $\texttt{Dis}_{ap}$ tend to zero, α play a role of margin, to force $\texttt{Dis}_{an} \geq \alpha$. α is usually called the *margin*

Contrastive Loss

- For any positive pair of data-points $y_{ij} = 1$ this distance should be small, and for negative pair $y_{ij} = 0$ it should be large.
- Contrastive loss:
  $L(\mathbf{x}_i, \mathbf{x}_j) = y_{ij}\mathtt{Dis}_{ij}^2 + (1 - y_{ij})\max(\alpha - \mathtt{Dis}_{ij}, 0)^2$
- Schroff, Florian, et al. Facenet: A unified embedding for face recognition and clustering, CVPR 2015.

Hard negative sampling

- After a few epochs, If (a, p, n) are chosen randomly, it will be easy to satisfy the inequality.



- Random sampling is inefficient to find these hard cases.

Hard negative sampling

Based on their distances, there are three categories of triplets:

1. Easy triplets, have a loss of 0 (and gradient equal to 0), because $\texttt{Dis}(a, p) + \alpha < \texttt{Dis}(a, n)$.
2. Hard triplets, where the negative is closer to thef anchor than the positive, $\texttt{Dis}(a, n) < \texttt{Dis}(a, p)$,
3. Semi-hard triplets, where negative pair is not closer to the anchor than the positive, but which still have positive loss: $(\texttt{Dis}(a, p) < \texttt{Dis}(a, n) < \texttt{Dis}(a, p) + \alpha)$

Smart Mining for Deep Metric Learning, B. Harwood et al, ICCV 2017

Triplet Rank Loss

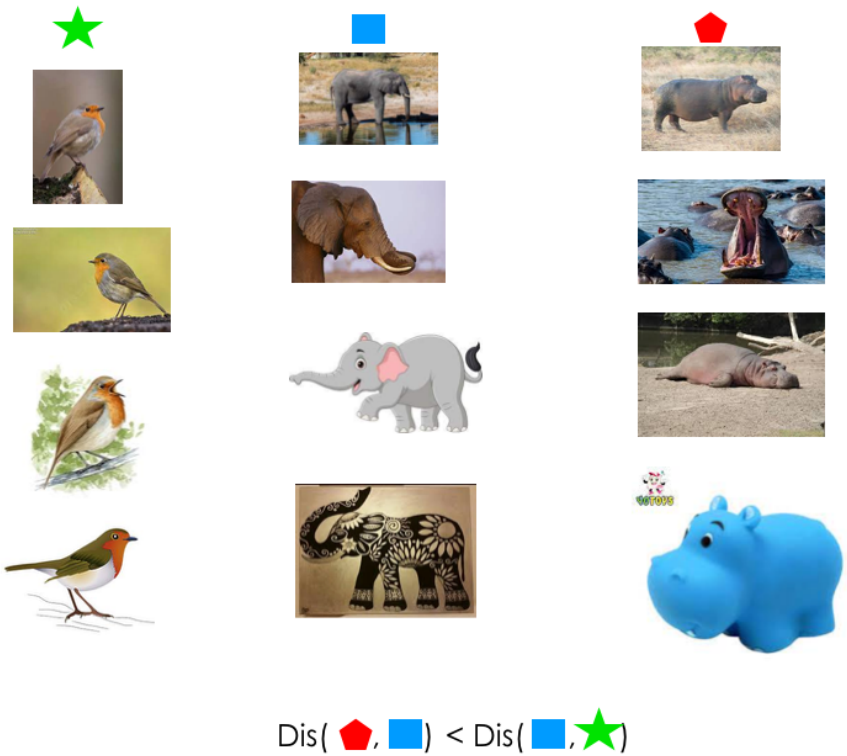


Figure: Rank between classes

- In general, one can define metric to consider relationship between classes.

Histogram loss

- Compute all the positives similarities and negatives similarities within a batch.
- The loss pushes to separate the two similarity distributions.

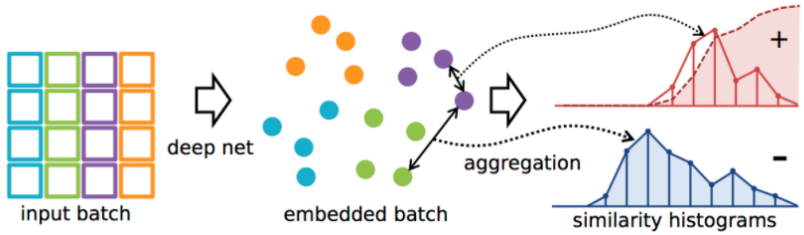


Figure: Histogram loss

Ustinova, Evgeniya, et al. Learning deep embeddings with histogram loss, NIPS 2016

Metric Learning in a nutshell

1. Pick a parametric distance $\texttt{Dis}(\cdot, \cdot)$
2. Collect similarity judgements on data pairs/triples/...
   - $\mathcal{S} = \{(\mathbf{x}_i, \mathbf{x}_j) : \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ are similar}\}$
   - $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{x}_j) : \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ are dissimilar}\}$
   - $\mathcal{R} = \{(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k) : \mathbf{x}_i \text{ is more similar to } \mathbf{x}_j \text{ than to } \mathbf{x}_k\}$
   - Training your model $f_{\boldsymbol{\theta}}$, to minimise a loss function:

$$\arg\min_{\boldsymbol{\theta}} L(\mathcal{S}, \mathcal{D}, \mathcal{R}) + \lambda \texttt{reg}(\boldsymbol{\theta}) \tag{1}$$

Deep Metric Learning: Second Approach

For a database of c types of objects (individuals)

1. Solve the metric learning problem as a classification problem of c classes with a final of softmax layer. In testing the score of similarity is a compute in the feature space before the use of softmax.
2. Advantage: Easy use of pretrained networks. You shouldn't bother with sampling triplets.
3. Disadvantage: Modification of softmax are required. Score is not directly optimized.

Deep Metric Learning: Second Approach

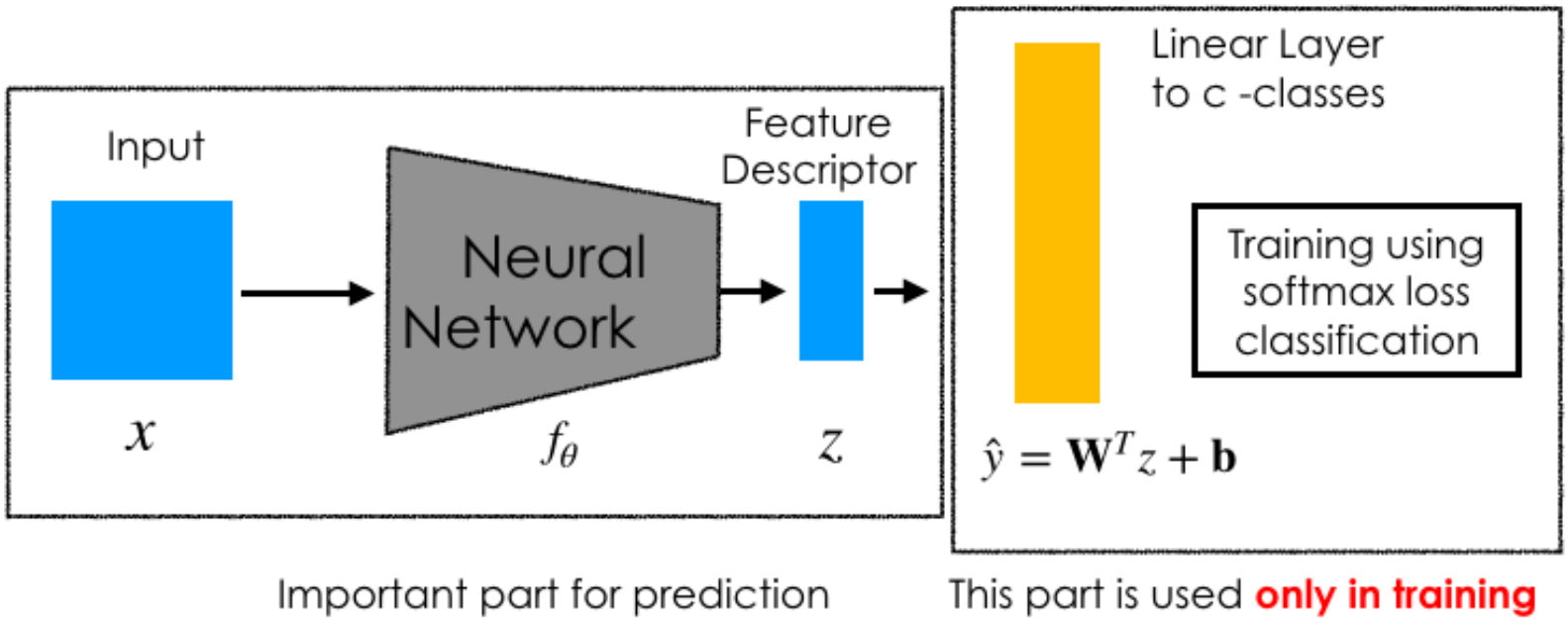


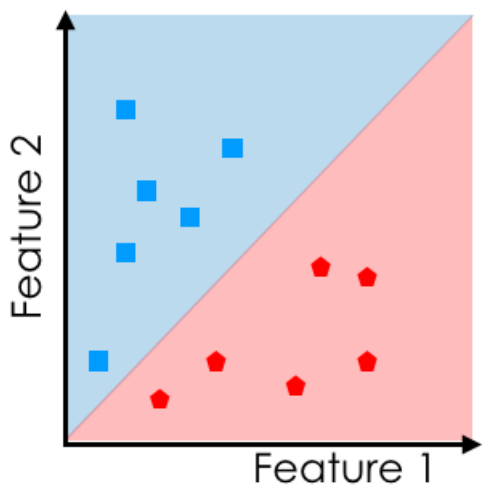
Figure: Metric Learning by means of Supervised Learning

Softmax

Let $z = f_{\boldsymbol{\theta}}(x)$ be the feature vector of the sample x after passing through the model $f_{\boldsymbol{\theta}}$. In the classification setting of c classes, we usually have a linear classification layer $\hat{y} = \mathbf{W}^T z + \mathbf{b}$, where $\mathbf{W} \in R^{n \times m}$ and $\mathbf{b} \in R^m$.

The Softmax Loss for a batch of N samples is defined as:

$$L_{softmax} = -\frac{1}{N} \sum_{i=1}^{N} \log \frac{\exp\left(\mathbf{W}_{y_i}^T z_i + \mathbf{b}_{y_i}\right)}{\sum_{j=1}^{c} \exp\left(\mathbf{W}_j^T z_i + \mathbf{b}_j\right)} \tag{2}$$

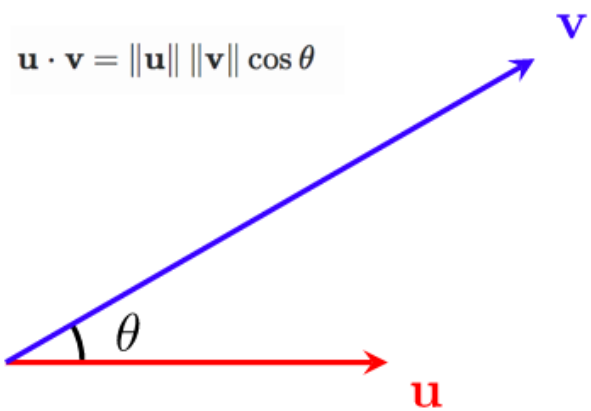


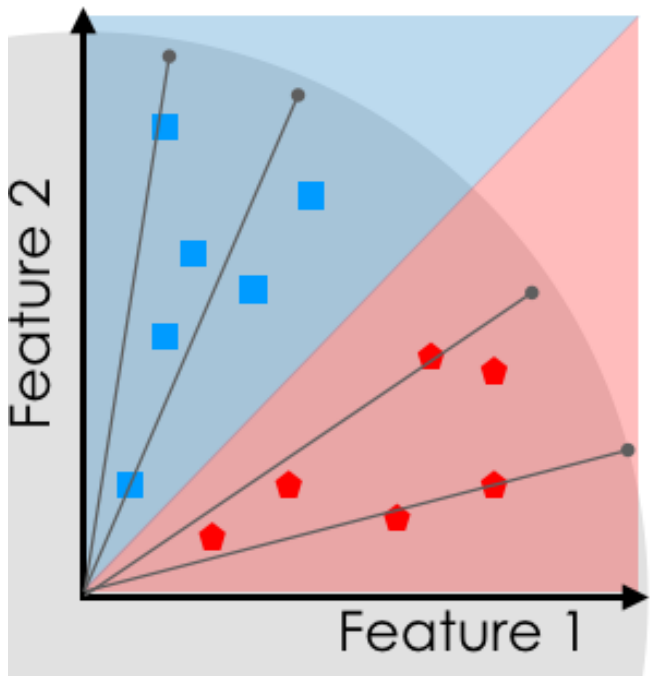
Modified Softmax

1. Set bias in the last layer to zero, $\mathbf{b} = 0$
2. Normalize the weight so that $||\mathbf{W}_j|| = 1$

then,

$$\exp\left(\mathbf{W}_j^T z_i + \mathbf{b}_j\right) = \exp(||\mathbf{W}_j||||z_i|| \cos(\theta_{j,i})) = \exp(||z_i|| \cos(\theta_{j,i}))$$





Modified Softmax

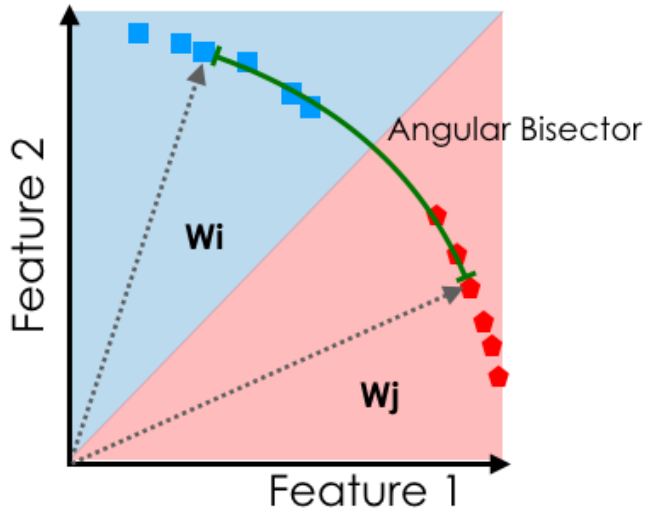


Figure: Modified SoftMax

Decision boundary is harder to make our features discriminative enough.

$$L_{ModSmax} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{\exp(||z_i||\cos(\theta_{y_i,i}))}{\sum_{j=1}^{c}\exp(||z_i||\cos(\theta_{j,i}))} \tag{3}$$

Geometrically, it means that we assign the sample to class k if the angle between $\mathbf{W}_k$ and z is the smallest among all class center $\mathbf{W}_j$.

CosFace

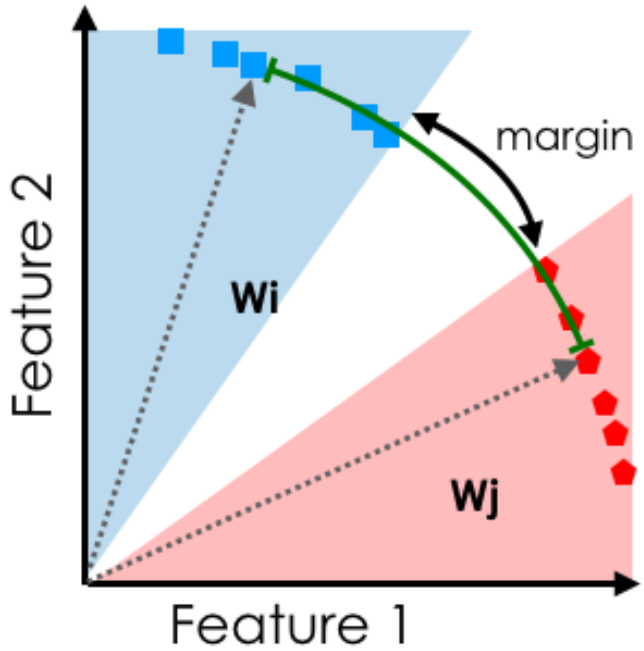


Figure: CosFace (Wang et al. 2018)

Add a normalization for the features z, so $||z|| = 1$. Add an *scaling* parameter s, and a *margin* parameter m.

$$L_{CosFace} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{\exp(s(\cos(\theta_{y_i,i}) - m))}{\exp(s\cos(\theta_{y_i,i}) - m) + \sum_{j\neq y_i}\exp(s\cos(\theta_{i,j}))} \tag{4}$$

Example

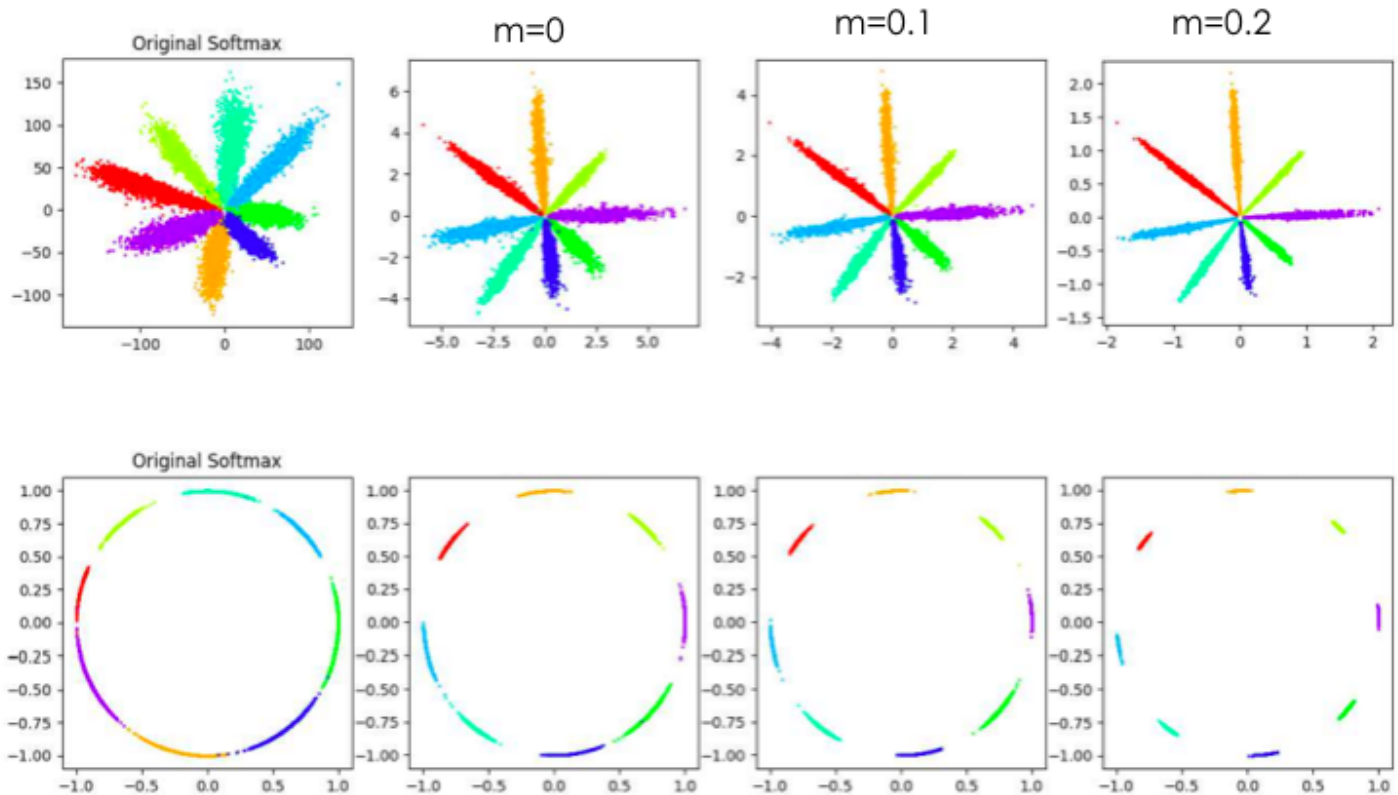


Figure: A toy experiment of different loss functions on eight identities with 2D features. The first row maps the 2D features onto the Euclidean space, while the second row projects the 2D features onto the angular space. The gap becomes evident as the margin term m increases. (Wang et al. 2018)

Practical Session

Contents

Re-identification

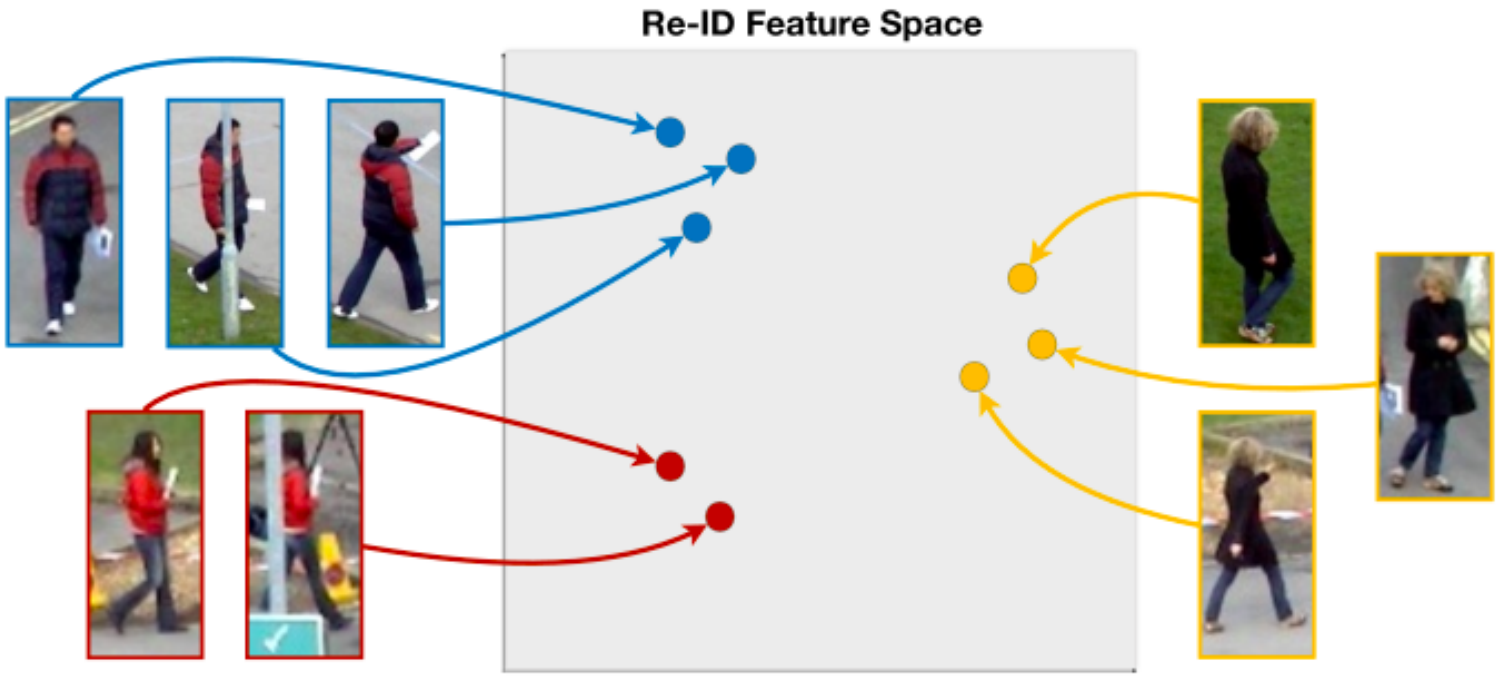


Figure: Person Re-identification

Example Similarity Learning Applications: Similar geometric reliefs

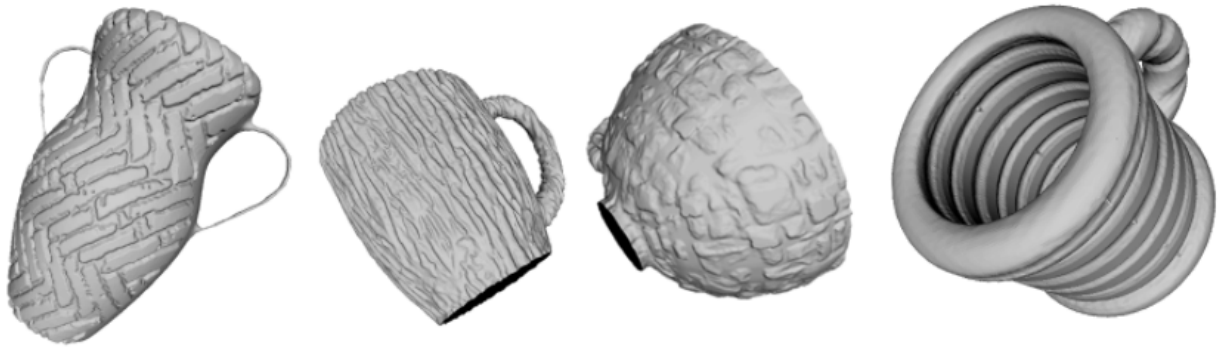


Figure: 220 surfaces characterized by different relief patterns. Only 3D shapes are availables.

SHREC'20 track: Retrieval of digital surfaces with similar geometric reliefs,E. Thompson et al, Computers and Graphics, 2020

Example Metric Learning Applications: Similar geometric reliefs

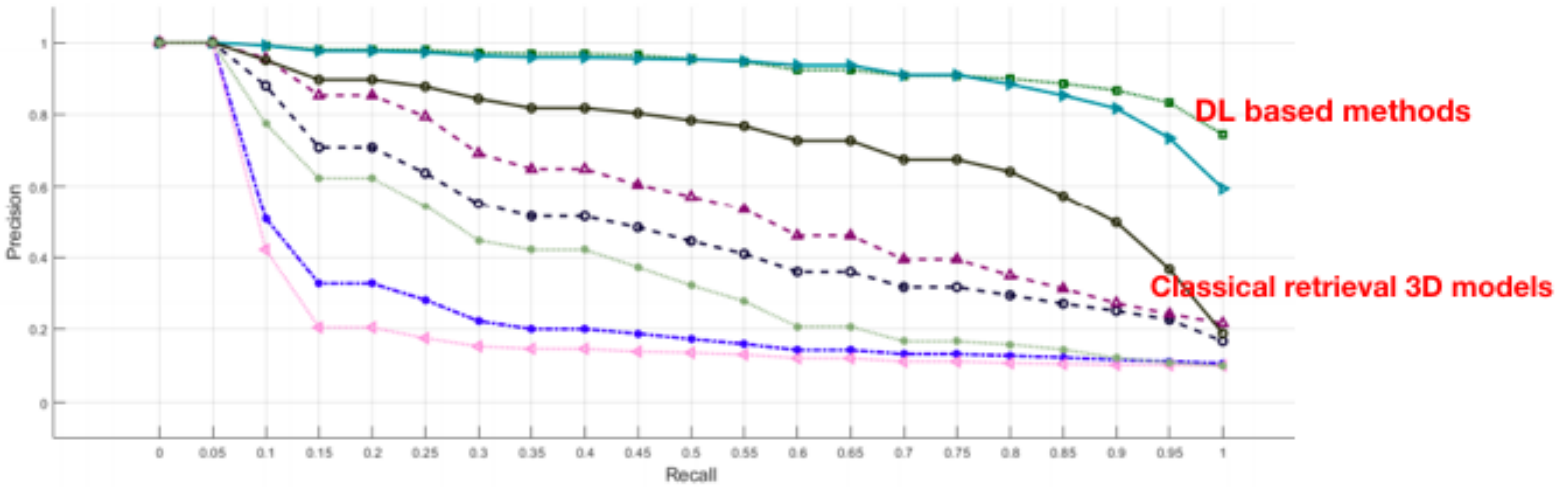


Figure: Results of different methods

Example Metric Learning Applications: Similar geometric reliefs

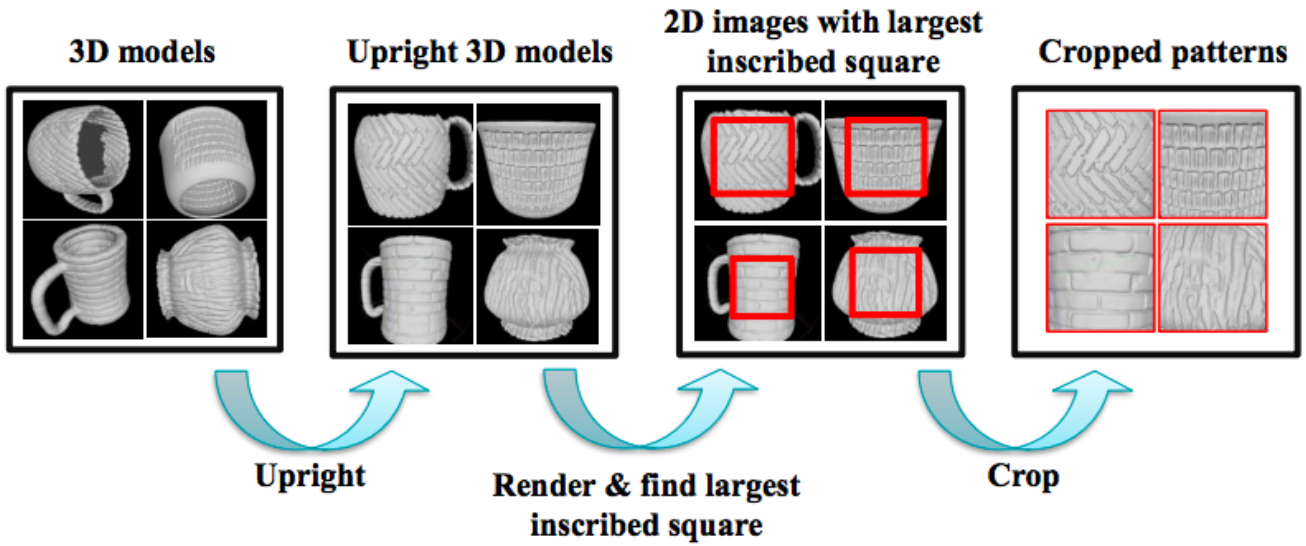


Figure: Using Triplet Loss on pretrained networks from automatic 3D to image mapping.

Other applications: Multimodal embeddings

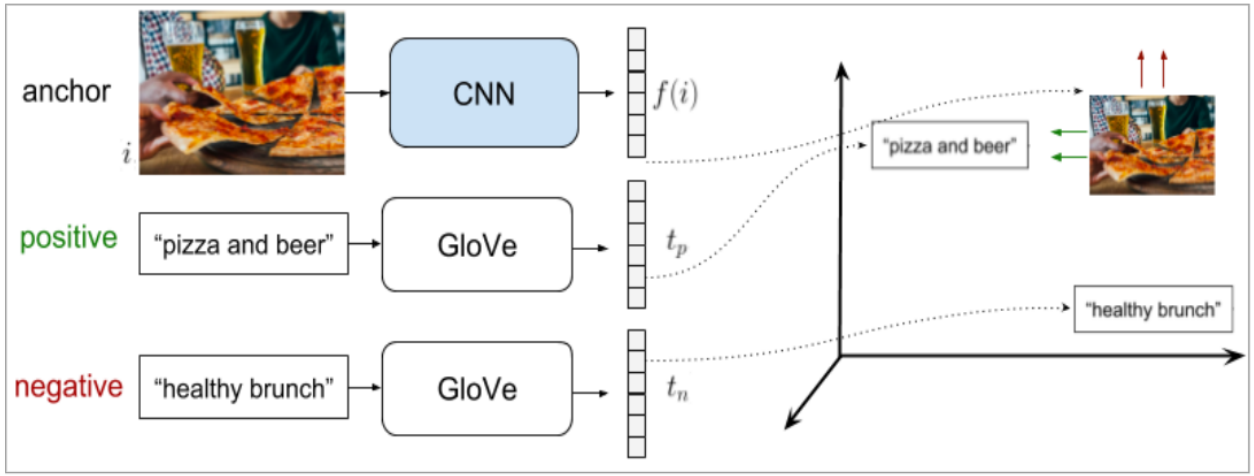


Figure: Multimodal embeddings

R. Gomez *et al.*,Learning to Learn from Web Data through Deep Semantic Embeddings, ECCV 2018.

Other applications: Multimodal embeddings

Figure: Multimodal embeddings

R. Gomez *et al.*,Learning to Learn from Web Data through Deep Semantic Embeddings, ECCV 2018.

Other applications: Multimodal embeddings

Figure: Multimodal embeddings

R. Gomez et al.,Learning to Learn from Web Data through Deep Semantic Embeddings, ECCV 2018.

Questions?

1. What is metric learning?
2. Why do you need to train with positive and negative examples? (Neural Collapse)
3. In which kind of problems Metric Learning can be useful?
4. How can the softmax be modified to induce greater separability in the classes?
5. How can supervised learning methods be used for the metric learning problem?

Contents

Adversarial Examples [Goodfellow et al., 2015]

- Given a ML model $f_{\boldsymbol{\theta}}(\cdot)$ 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 $f_{\boldsymbol{\theta}}(\mathbf{x}) \neq f_{\boldsymbol{\theta}}(\mathbf{x}^{adv})$ 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 examples for benign samples $\mathbf{x}$, i.e., samples where the model gives a correct prediction.

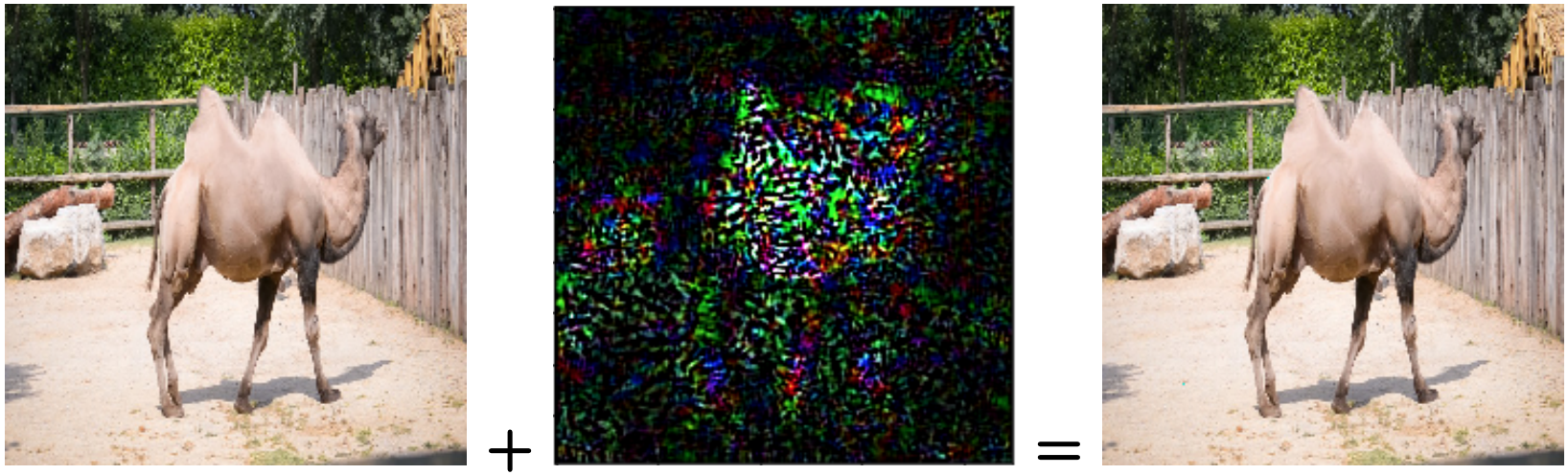


Figure: $\mathbf{x} + \epsilon = \mathbf{x}^{adv}$

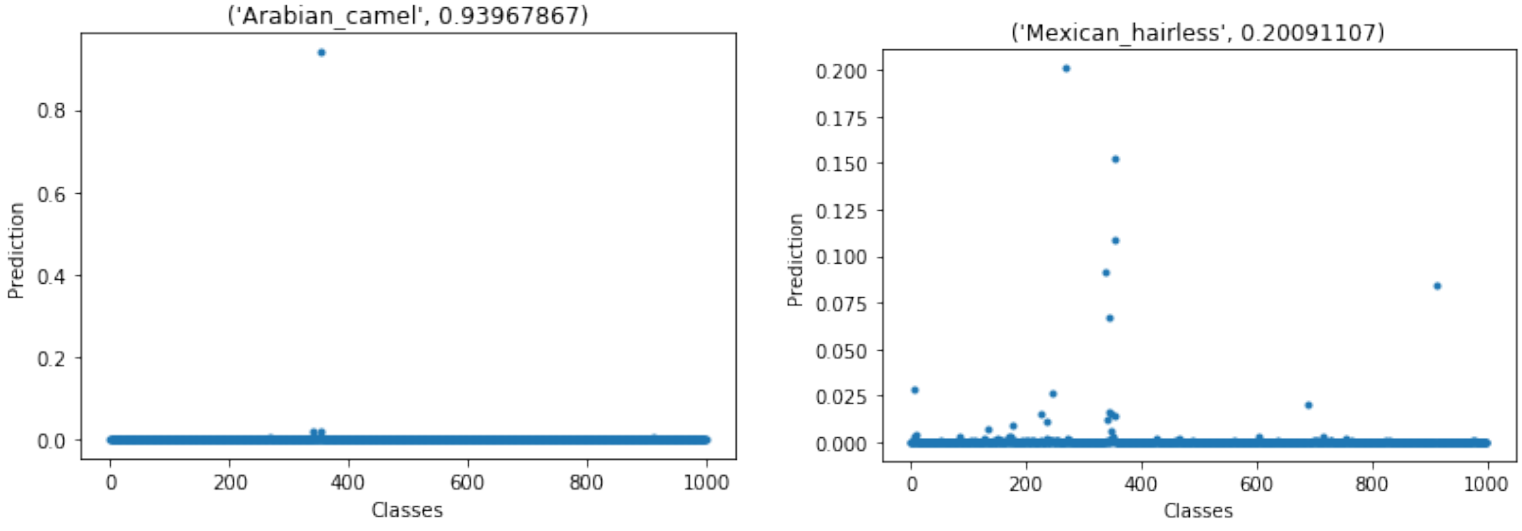


Figure: VGG19 Prediction for: $\mathbf{x}$ and $\mathbf{x}^{adv}$

Adversarial Gradient

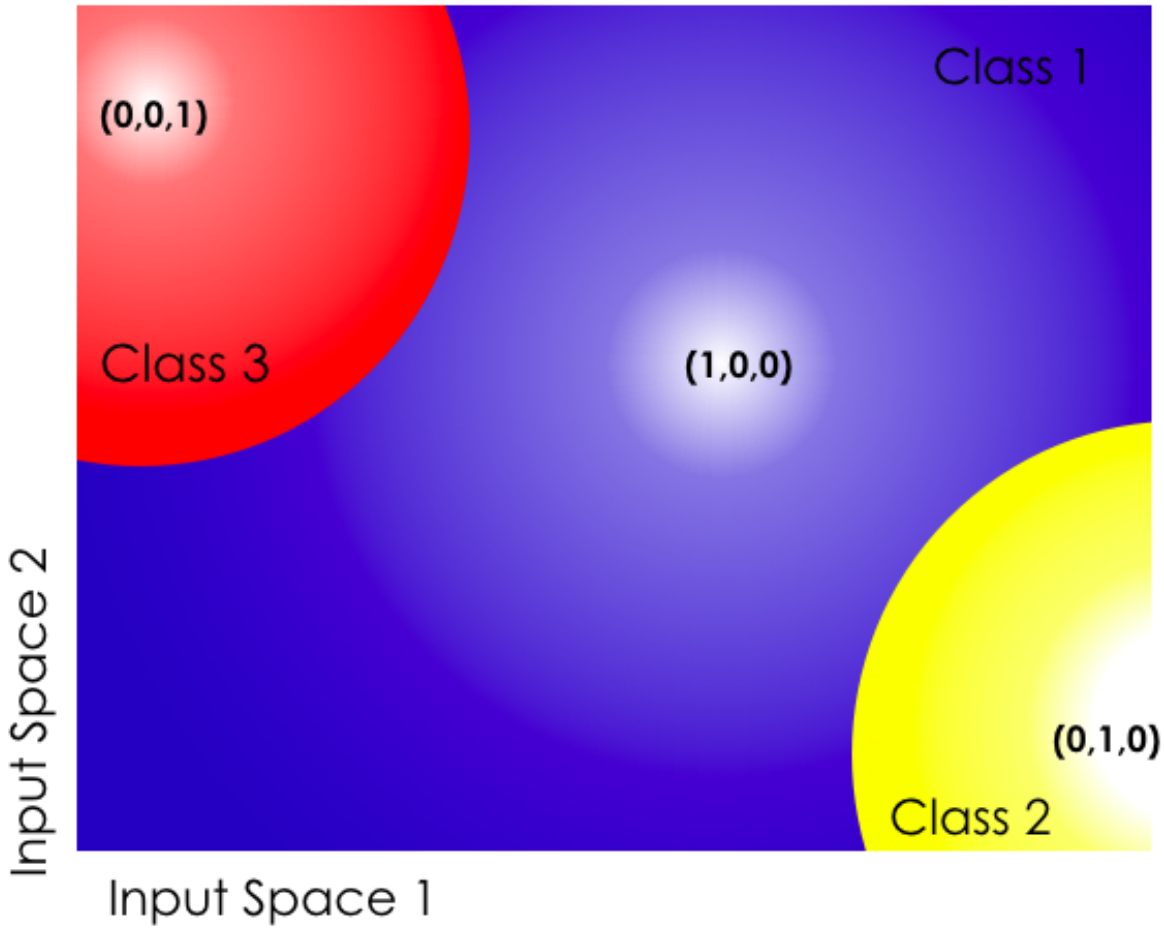


Figure: Figure shows the prediction of the model for three points in the input space. Additionally, colours represent the class predicted by the model.

Adversarial Gradient

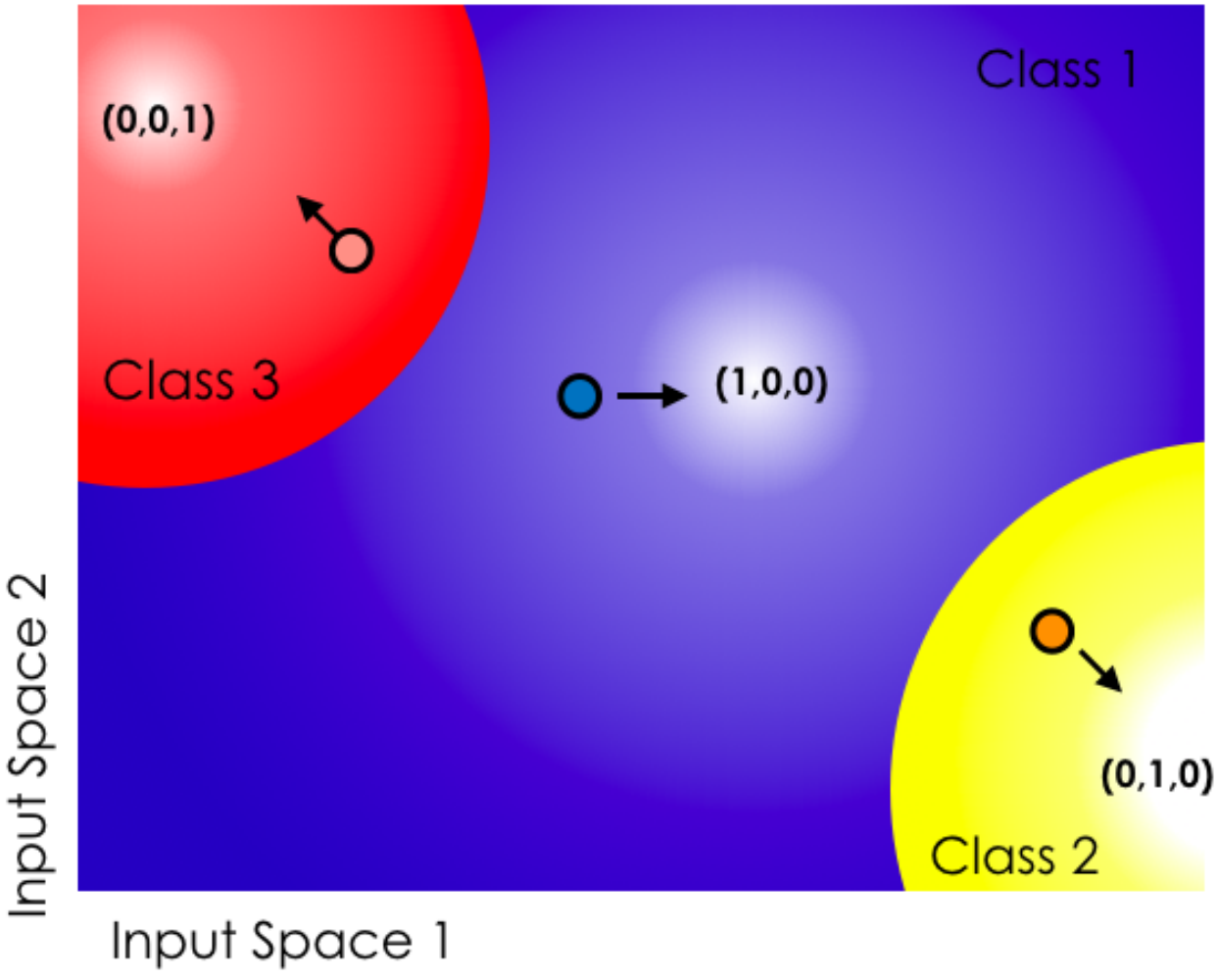


Figure: $-\nabla_{\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), y_{\mathtt{true}}))$

Adversarial Gradient

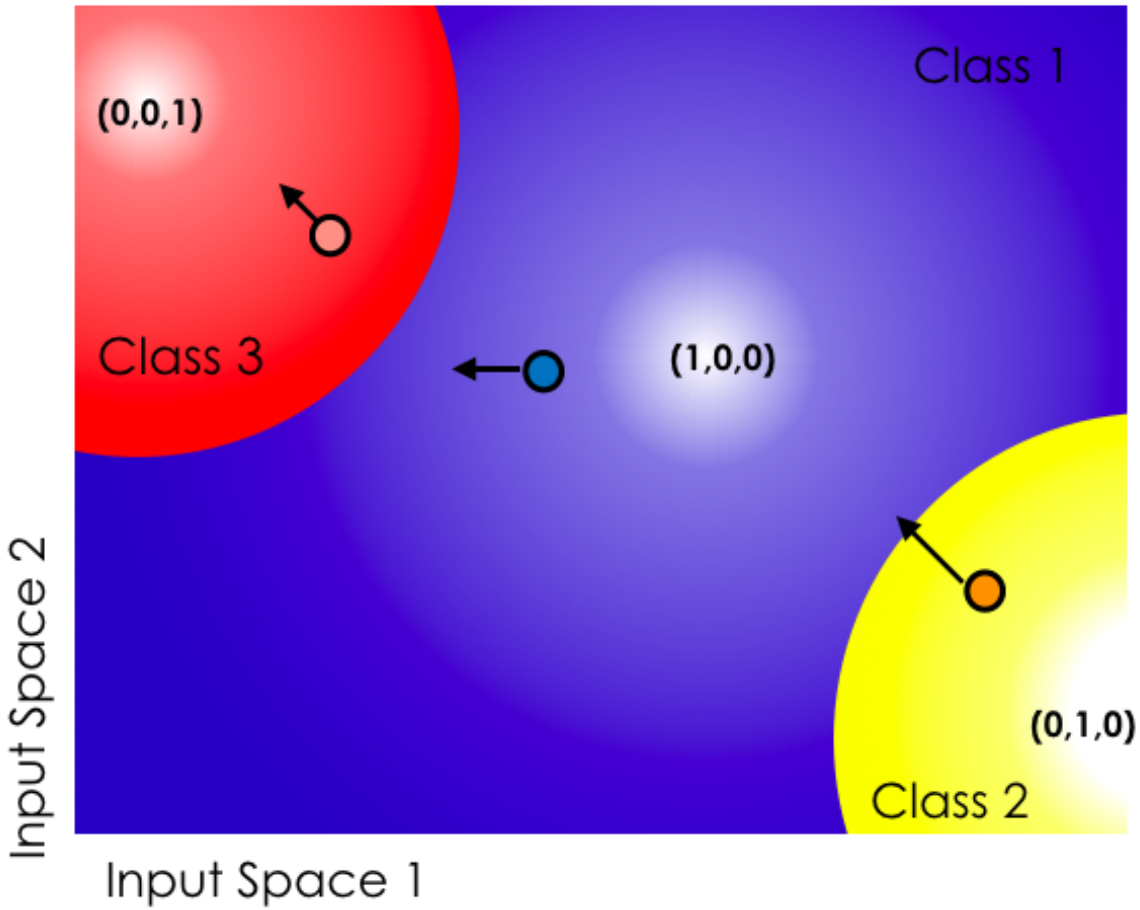


Figure: $-\nabla_{\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), (0,0,1)))$

Gradient based Methods

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

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

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.
- This is a single-step, untargeted attack, which approximately minimises the L_∞ norm of the perturbation bounded by the parameter ϵ.

Gradient based Methods

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

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

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.
- This is a single-step, untargeted attack, which approximately minimises the L_∞ norm of the perturbation bounded by the parameter ϵ.

Adversarial examples as data augmentation

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

Fast Gradient Sign Method II

This single-step attack encourages the network to classify the adversarial example as $y_{\mathtt{t}}$ by assigning

$$\mathbf{x}^{adv} = \mathbf{x} - \epsilon \cdot sign(\nabla_{\mathbf{x}} L(f_{\boldsymbol{\theta}}(\mathbf{x},), y_{\mathtt{t}})) \tag{6}$$

In reference, the convention is choosing the target class as the least likely class predicted by the network.

Iterative FGSM

This attack extends FGSM by applying it in an iterative manner, which increases the chance of fooling the original network. Using the subscript to denote the iteration number, this can be written as

$$\mathbf{x}_0^{adv} = \mathbf{x}$$

$$\mathbf{x}_{t+1}^{adv} = \texttt{clip}(\mathbf{x}_t^{adv} + \alpha \cdot sign(\nabla_{\mathbf{x}_t^{adv}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), y_{\mathtt{t}}))$$

The $\texttt{clip}(\mathbf{a})$ makes each value on a_i of $\mathbf{a}$ is in range $[a_i - \epsilon, a_i + \epsilon]$. This ensures that the max-norm constraint of each component of the perturbation, being no greater than ϵ is maintained. It thus corresponds to projected gradient descent **??**, with step-size ∇, into an L_∞ ball of radius ϵ around the input $\mathbf{x}$.

Universal adversarial perturbation [Moosavi-Dezfooli et al., 2017]

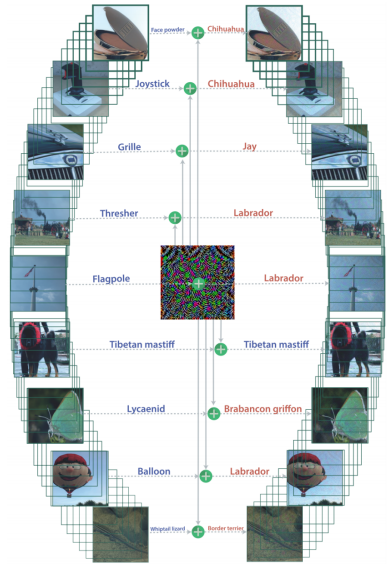


Figure: When added to a natural image, an universal perturbation image causes the image to be misclassified by the deep neural network with high probability

Available code in:
`https://github.com/KhrulkovV/singular-fool`

White box Attacks

Without having access to model's parameters, but only prediction value, some works show that is possible to create adversarial examples by free-derivate methods.

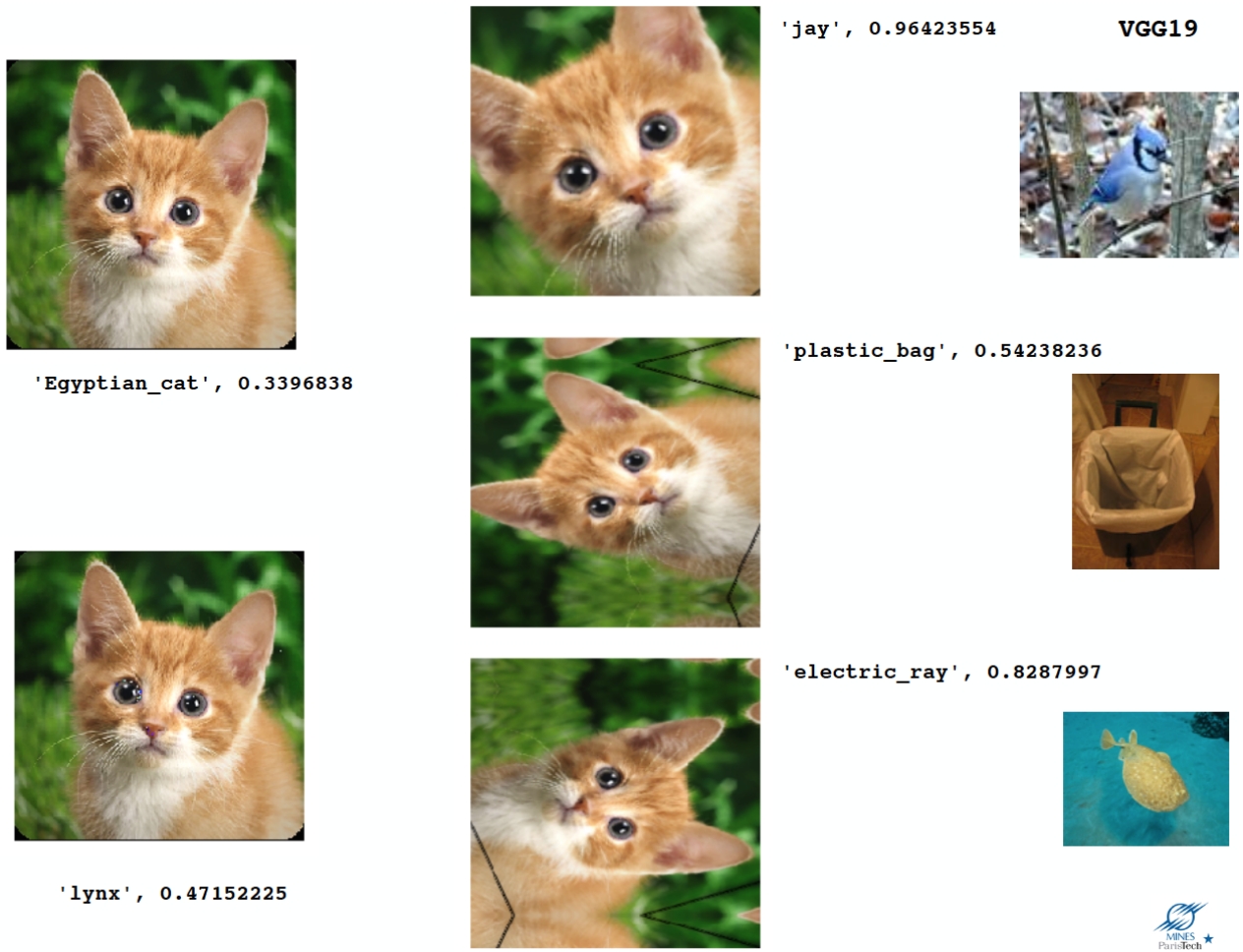


Figure: [Akhtar and Mian, 2018]

One-pixel attack

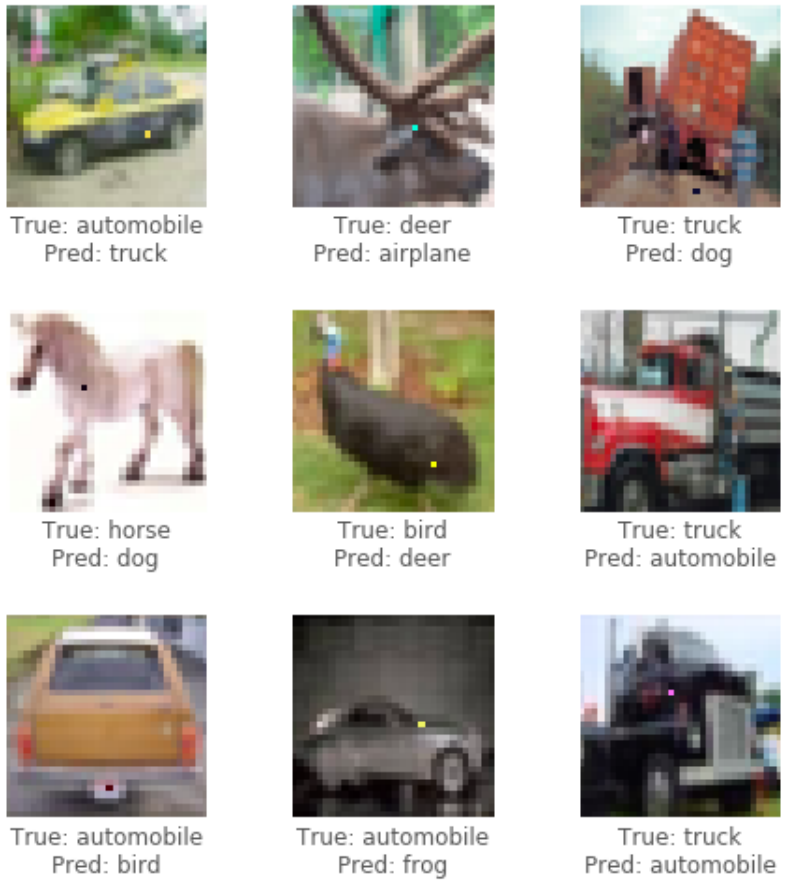


Figure: [Su et al., 2019]

Available code in:
`https://github.com/Hyperparticle/one-pixel-attack-keras`

Interest of adversarial attacks

- `https://foolbox.readthedocs.io/en/latest/index.html`
- Improve our understanding of deep learning
- Build defences against these attacks
- Improve the robustness of the models
- Reduce the overfitting during training.

Practical Session: Adversarial attacks

https://drive.google.com/file/d/
1bE5Tyo-p7DF9Q-VDHiEjRNrlq9ABdrCj/view?usp=sharing

Questions?

1. What is an adversarial example?
2. Can adversarial examples helps to avoid overfitting during training?
3. Could you explain a technique to create adversarial examples?

Contents

Goal

Generative models "learn" a joint distribution over a dataset. They are mostly used for sampling applications or density estimation.

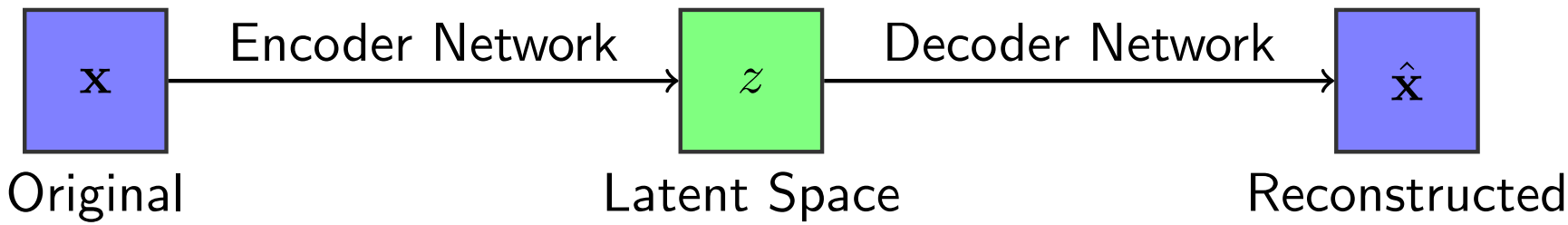
1. (Sampling) A generative model learns to fit a model over observations so one can sample novel data from the model, $\mathbf{x}_{new} \sim p_{\boldsymbol{\theta}}(\mathbf{x})$
2. (Density estimation) A model learns to estimate the probability of observations. Given a datapoint $\mathbf{x}$ the model, what is the probability of $\mathbf{x}$ belong to the dataset?

Contents

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.



The loss function for a given input vector is usually the reconstruction error:

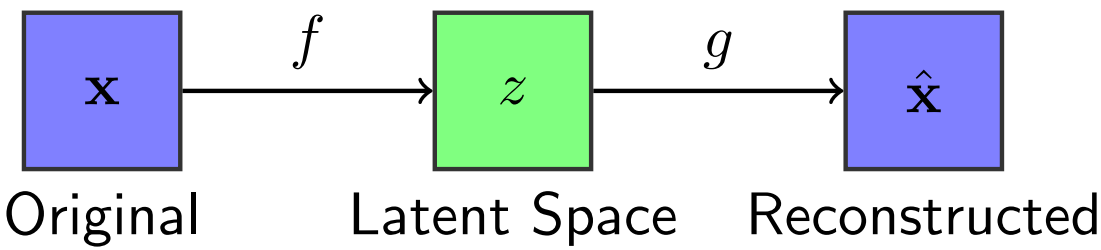
$$L(\mathbf{x}) = \|\mathbf{x} - \hat{\mathbf{x}}\|^2$$

Autoencoder

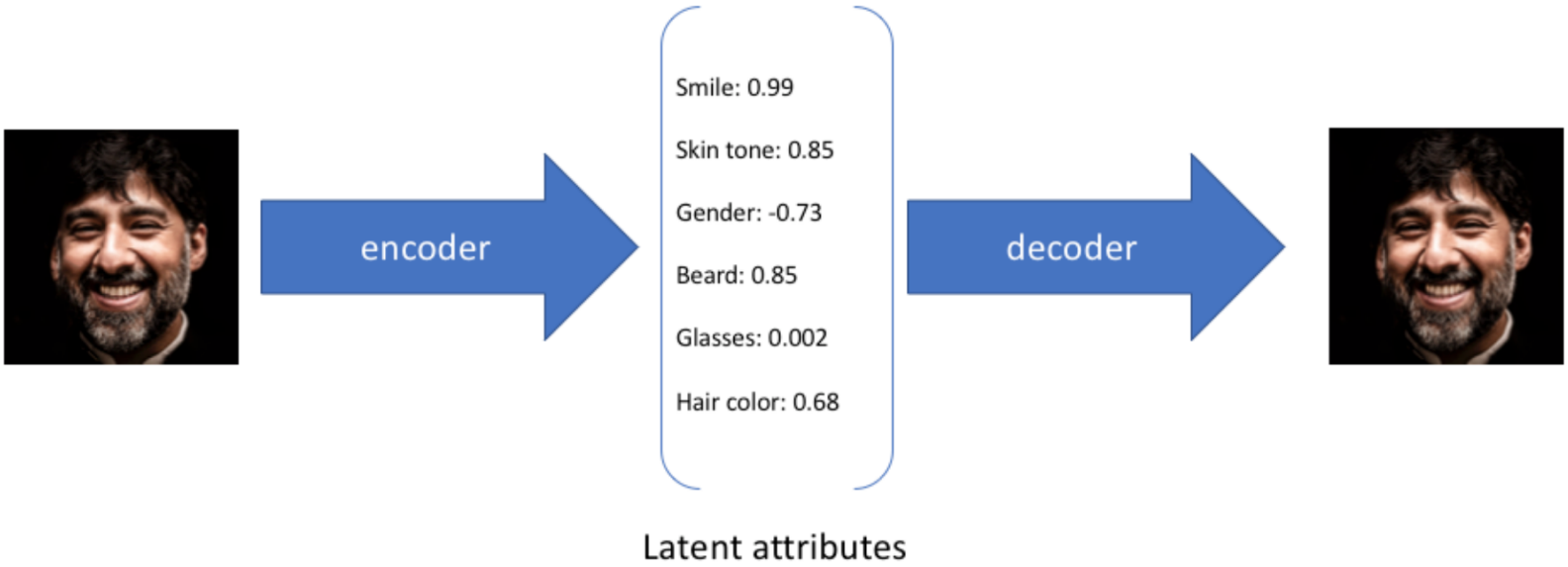
- An *autoencoder* is a neural network that is trained to attempt to copy its input to its output.
- The network may be viewed as consisting of two parts: an encoder function $z = f(\mathbf{x})$ and a decoder that produces a reconstruction $r(\mathbf{x}) = g(f(\mathbf{x}))$.
- The composition of f and 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 consists in minimizing the loss function:

$$L(\mathbf{x}, g(f(\mathbf{x}))), \tag{7}$$

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



Latent space intuition



Credits:
https://www.jeremyjordan.me/variational-autoencoders/

Type of Autoencoders:

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

Regularized Autoencoders

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

$$L(\mathbf{x}, g(f(\mathbf{x}))) + \Omega(f) \tag{8}$$

- L_1: $\Omega(f) = \lambda \sum ||\boldsymbol{\theta}||$
- L_2: $\Omega(f) = \lambda \sum ||\boldsymbol{\theta}||^2$

```
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.

Example: L_1 sparcity constraint

$$L(\mathbf{x}, g(f(\mathbf{x}))) + \lambda \sum_i |\boldsymbol{\theta}_i|$$

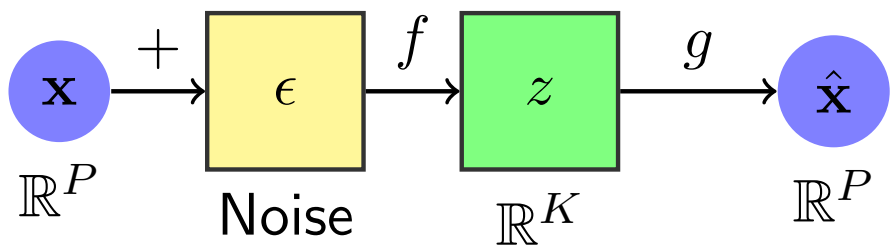
Denoising Autoencoders (DAE)[Vincent et al., 2008]

Idea

Add noise to the inputs to avoid overfitting. The loss is then:

$$L(\mathbf{x}, g(f(\tilde{\mathbf{x}}))), \tag{9}$$

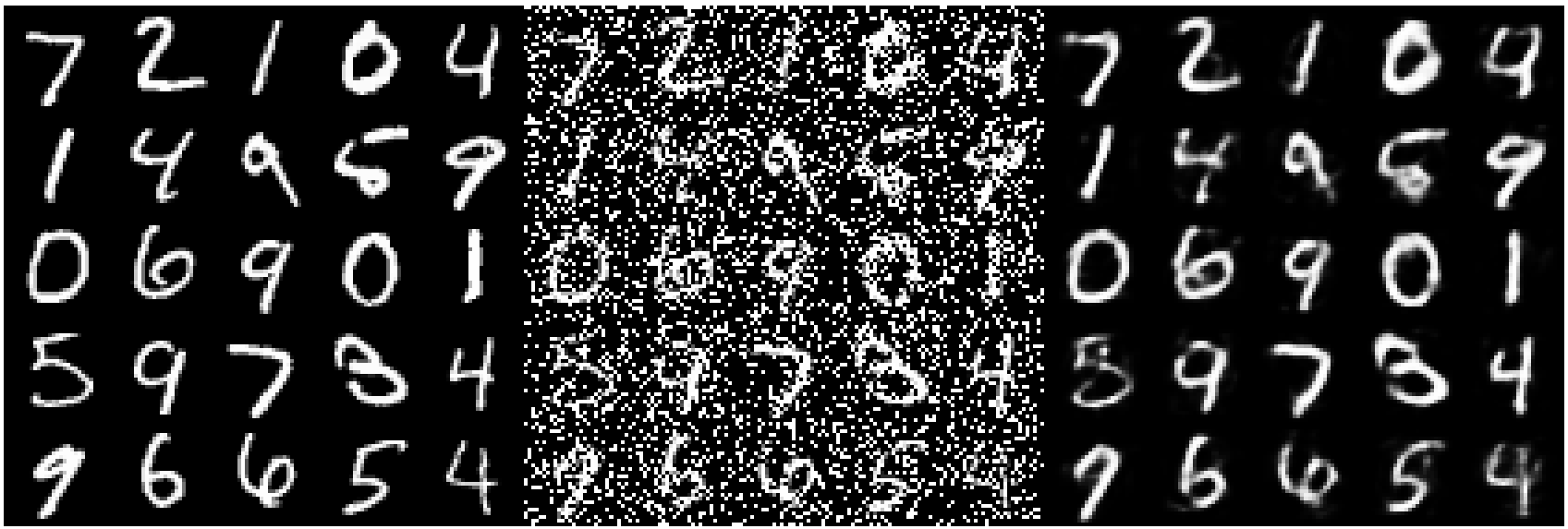
where $\tilde{\mathbf{x}}$ is a corrupted copy of $\mathbf{x}$ by some form of noise.



An over-complete autoencoder with high capacity can end up learning an identity function where input=output. Add noise to avoid overfitting.

DAE example

In [Vincent et al., 2008] the noise consists in setting to zero each pixel with a given probability p. Typical values are 0.3 or 0.5.



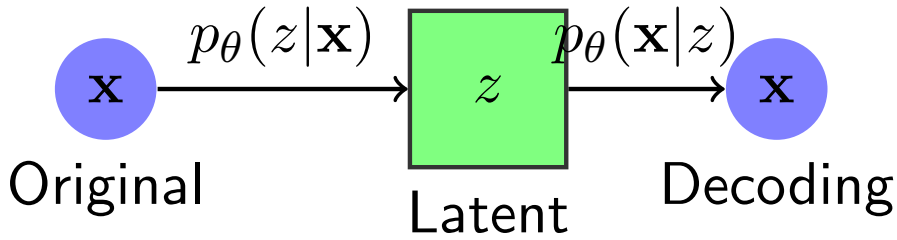
Credits:
http://www.opendeep.org/v0.0.5/docs/tutc
your-first-model

Contents

Modern Autoencoder

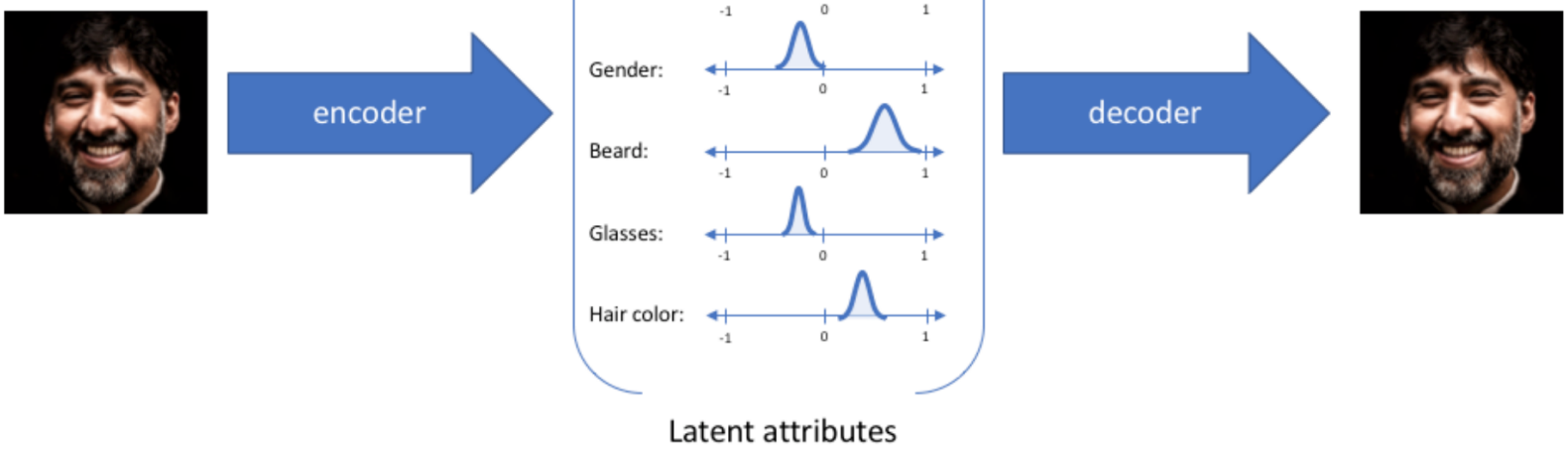
- Modern autoencoders have generalized the idea of an encoder and a decoder beyond deterministic functions to stochastic mappings $p_{\texttt{encoder}}(z|\mathbf{x})$ and $p_{\texttt{decoder}}(\mathbf{x}|z)$.

Variational Autoencoder



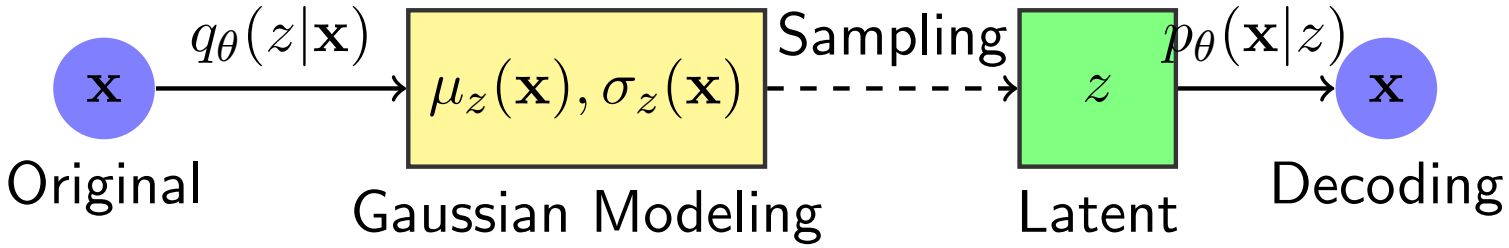
- Training via maximum likelihood of $p(\mathbf{x})$
- Intractability: the true posterior density $p_\theta(z|\mathbf{x})$ cannot be calculated in polynomial time
- Solution: Approximate $p_\theta(z|\mathbf{x})$ by means of $q_\theta(z|\mathbf{x}) = \mathcal{N}(z; \mu_z(x), \sigma_z(x))$

Latent space intuition (variational case)



Credits:
https://www.jeremyjordan.me/variational-autoencoders/

Variational Autoencoder



- Gaussian modeling
- Training via maximum likelihood of $p(\mathbf{x})$
- Learning the parameters θ via backpropagation?

Training via maximum likelihood

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

$$
\begin{aligned}
\mathcal{L}(\mathbf{x}) &= \log(p(\mathbf{x})) \\
&= \sum_z q(z|\mathbf{x}) \log(p(\mathbf{x})) \\
&= \sum_z q(z|\mathbf{x}) \log\left(\frac{p(z,\mathbf{x})}{p(z|\mathbf{x})}\right) \\
&= \sum_z q(z|\mathbf{x}) \log\left(\frac{p(z,\mathbf{x})q(z|\mathbf{x})}{q(z|\mathbf{x})p(z|\mathbf{x})}\right) \\
&= \sum_z q(z|\mathbf{x}) \log\left(\frac{p(z,\mathbf{x})}{q(z|\mathbf{x})}\right) + \sum_z q(z|\mathbf{x}) \log\left(\frac{q(z|\mathbf{x})}{p(z|\mathbf{x})}\right) \\
&= \sum_z q(z|\mathbf{x}) \log\left(\frac{p(z,\mathbf{x})}{q(z|\mathbf{x})}\right) + \underbrace{KL(q(z|\mathbf{x}), p(z|\mathbf{x}))}_{\geq 0} \\
&= \mathcal{L}^{lvb}(\mathbf{x}) + KL(q(z|\mathbf{x}), p(z|\mathbf{x})) \\
&\geq \mathcal{L}^{lvb}(\mathbf{x})
\end{aligned}
$$

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

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

Reparametrization trick

Backpropagation is possible through random sampling!

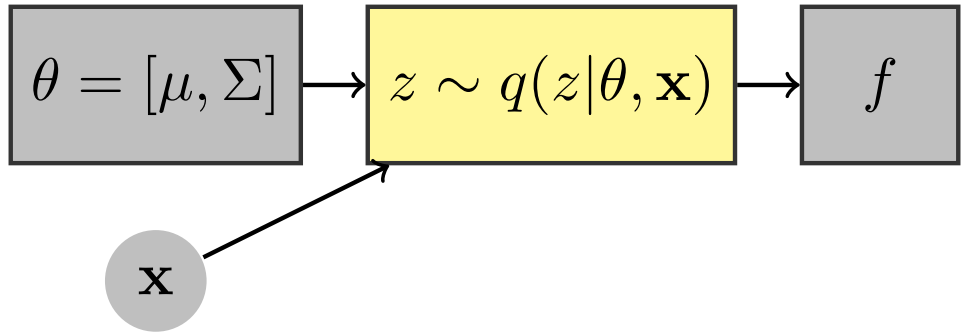


Figure: Original Formulation

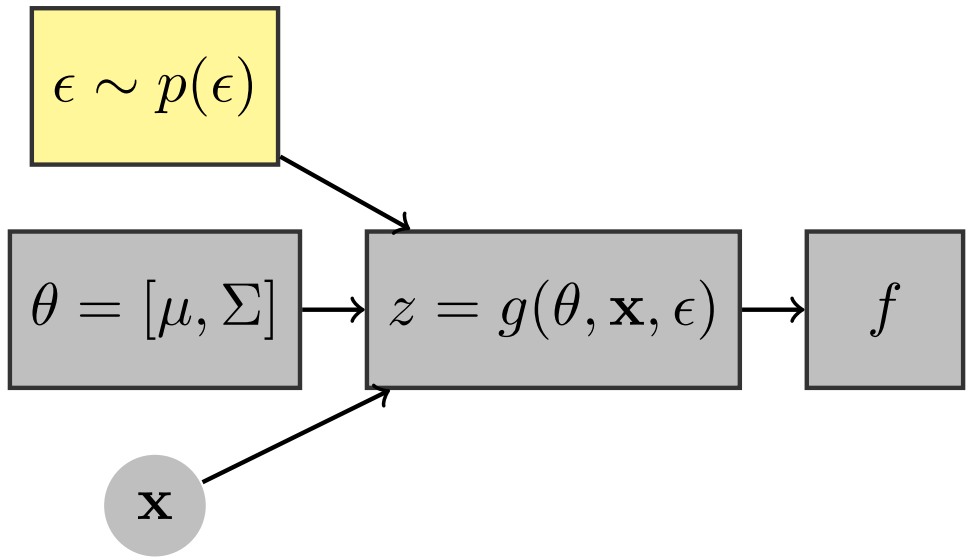


Figure: Reparametrization trick. Backpropagation

Applications of autoencoders

- Data denoising
- Dimensionality Reduction
- Database understanding
- Unsupervised pre-training (First self-training approach)
- Generative models

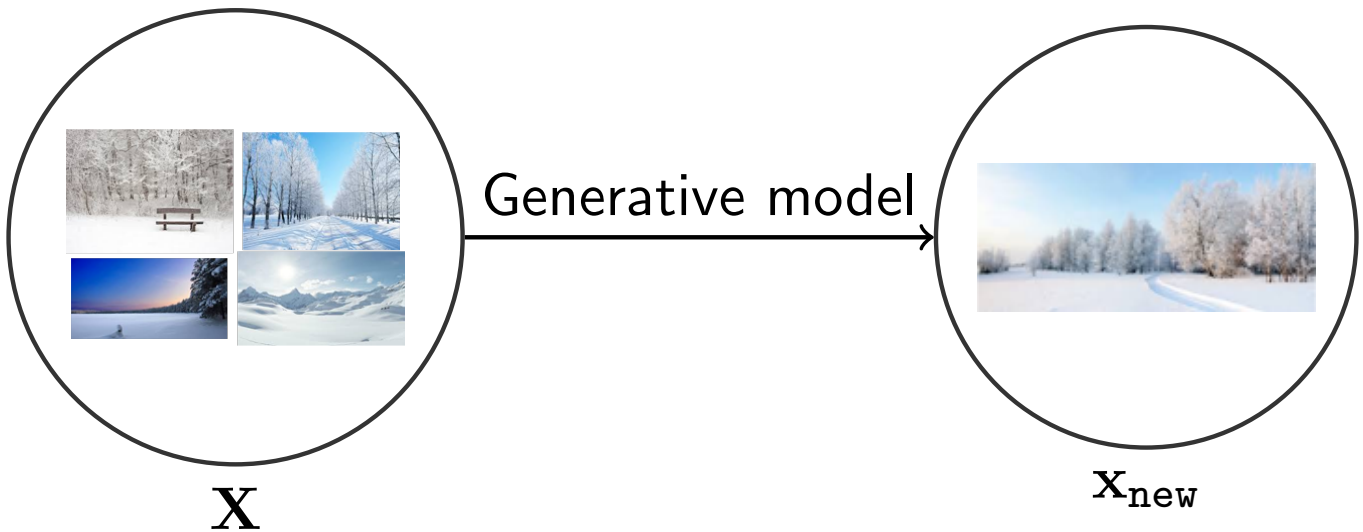
Contents

Which face is real?

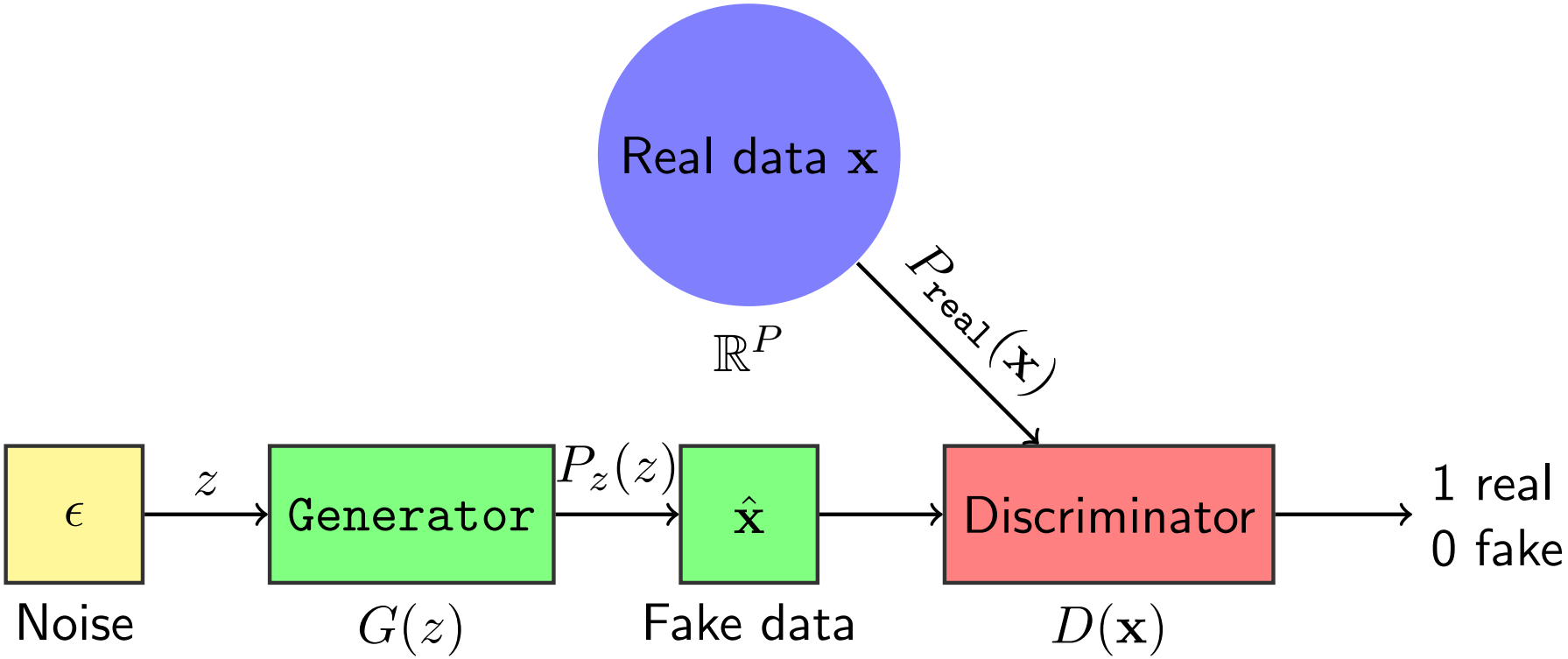
Generator available online from `http://www.whichfaceisreal.com/`, based on StyleGAN by Tero Karras, Samuli Laine, and Timo Aila from NVIDIA.

Unsupervised learning: Generative Models

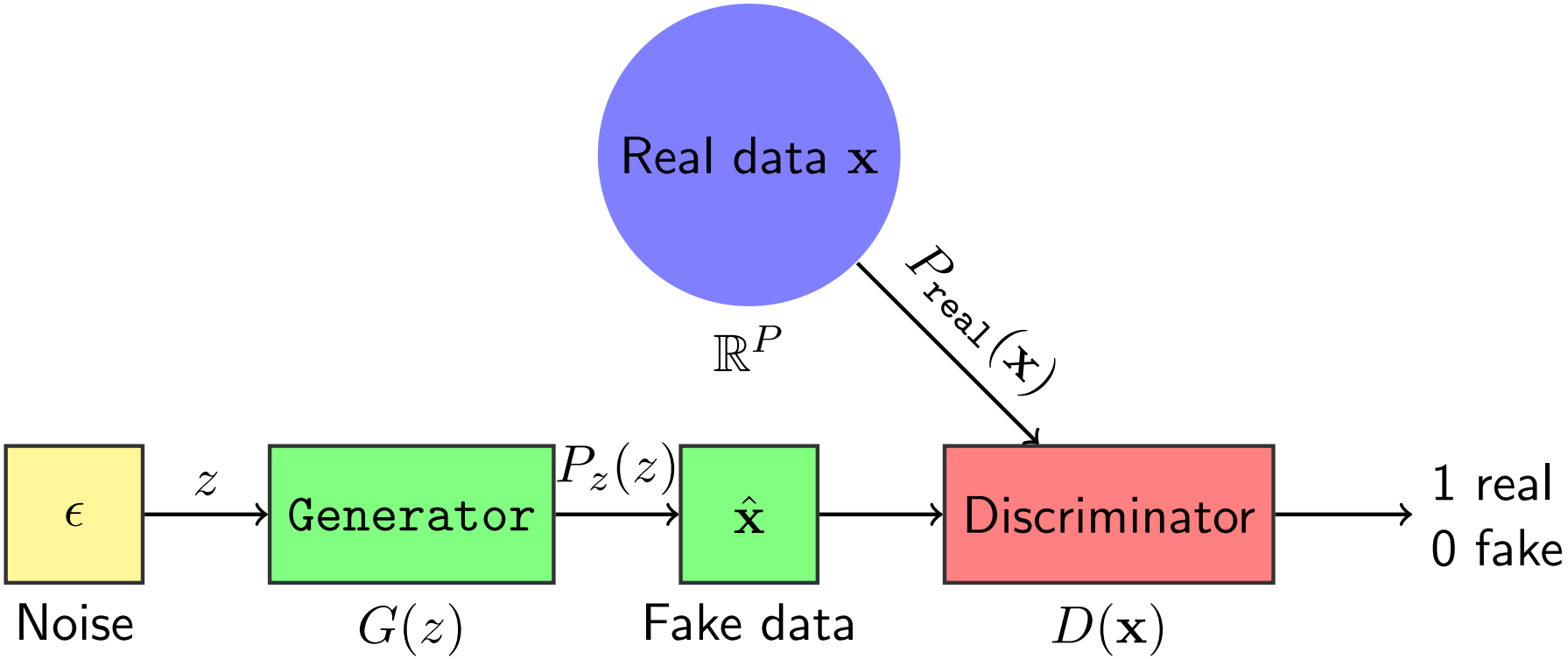
Given an unlabeled dataset $(\mathbf{X})$, we would like to learn: How to generate a new observations from the same distribution (unknown) of dataset?



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



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



"The coolest idea in machine learning in the last 20 years" - Yann LeCun.

Generative Adversarial Networks (GANs)

We require that the discriminator D recognises 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 trick 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:

$$\min_G \max_D V(G, D) \tag{10}$$

Generative Adversarial Networks (GANs)

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

$$V(G,D) = \int_x \left(P_{\texttt{real}}(\mathbf{x})[\log D(\mathbf{x})] + P_{g(\mathbf{x})}[\log(1 - D(x))] \right) dx \tag{11}$$

- Note that $f(x) = a\log(x) + b\log(1-x)$, has minimum in $x^* = \frac{a}{a+b}$

$$V(G,D) = \int_x \left(P_{\texttt{real}}(\mathbf{x})[\log D(\mathbf{x})] + P_{g(\mathbf{x})}[\log(1 - D(x))] \right) dx \tag{11}$$

- Note that $f(x) = a\log(x) + b\log(1-x)$, has minimum in $x^* = \frac{a}{a+b}$
- Then If G is fixed,
- $P_{\texttt{real}}(\mathbf{x}) \log D(\mathbf{x}) + P_{g(\mathbf{x})} \log(1 - D(x))$ the optimal discriminator $D^*(\mathbf{x}) = \frac{P_{\texttt{real}}(\mathbf{x})}{P_{\texttt{real}}(\mathbf{x}) + P_{\texttt{g}}(\mathbf{x})}$

Given an optimal discriminator $D^*(\mathbf{x})$, find $\min_G V(G, D^*)$.

$$\int_x (P_{\texttt{real}}(\mathbf{x}) \log\left(\frac{P_{\texttt{real}}(\mathbf{x})}{P_{\texttt{real}}(\mathbf{x}) + P_{\texttt{g}}(\mathbf{x})}\right) + P_{g(\mathbf{x})} \log\left(1 - \frac{P_{\texttt{real}}(\mathbf{x})}{P_{\texttt{real}}(\mathbf{x}) + P_{\texttt{g}}(\mathbf{x})}\right)$$

[2]Divergence Jensen–Shannon is a symmetrized and smoothed version of the Kullback–Leibler divergence, $Div_{\texttt{JS}}(p||q) = \frac{1}{2} Div_{\texttt{KL}}(p||m) + \frac{1}{2} Div_{\texttt{KL}}(q||m)$ where $m = p + q$

Given an optimal discriminator $D^*(\mathbf{x})$, find $\min_G V(G, D^*)$.

$$\int_x (P_{\texttt{real}}(\mathbf{x}) \log\left(\frac{P_{\texttt{real}}(\mathbf{x})}{P_{\texttt{real}}(\mathbf{x}) + P_{\texttt{g}}(\mathbf{x})}\right) + P_{g(\mathbf{x})} \log\left(1 - \frac{P_{\texttt{real}}(\mathbf{x})}{P_{\texttt{real}}(\mathbf{x}) + P_{\texttt{g}}(\mathbf{x})}\right)$$

by means of Divergence Jensen–Shannon[2] we can obtain:

$$Div_{\texttt{JS}}(P_{\texttt{real}} || P_{\texttt{g}}) = \frac{1}{2}(\log 4 + \min_G V(G, D^*)) \qquad (12)$$

[2]Divergence Jensen–Shannon is a symmetrized and smoothed version of the Kullback–Leibler divergence, $Div_{\texttt{JS}}(p||q) = \frac{1}{2} Div_{\texttt{KL}}(p||m) + \frac{1}{2} Div_{\texttt{KL}}(q||m)$ where $m = p + q$

Summarising, we have:

- Optimal Discriminator: $D^*(\mathbf{x}) = \frac{P_{\texttt{real}}(\mathbf{x})}{P_{\texttt{real}}(\mathbf{x}) + P_{\texttt{g}}(\mathbf{x})}$
- $\min_G V(G, D^*) = 2Div_{\texttt{JS}}(P_{\texttt{real}} || P_{\texttt{g}}) - 2\log 2$

Summarising, we have:

- Optimal Discriminator: $D^*(\mathbf{x}) = \frac{P_{\texttt{real}}(\mathbf{x})}{P_{\texttt{real}}(\mathbf{x}) + P_{\texttt{g}}(\mathbf{x})}$
- $\min_G V(G, D^*) = 2Div_{\texttt{JS}}(P_{\texttt{real}} || P_{\texttt{g}}) - 2\log 2$

Thus,

- $\min_G V(G, D^*)$ is obtained when $P_{\texttt{real}} = P_{\texttt{g}}$,

Summarising, we have:

- Optimal Discriminator: $D^*(\mathbf{x}) = \frac{P_{\texttt{real}}(\mathbf{x})}{P_{\texttt{real}}(\mathbf{x}) + P_{\texttt{g}}(\mathbf{x})}$
- $\min_G V(G, D^*) = 2Div_{\texttt{JS}}(P_{\texttt{real}} || P_{\texttt{g}}) - 2\log 2$

Thus,

- $\min_G V(G, D^*)$ is obtained when $P_{\texttt{real}} = P_{\texttt{g}}$,
- $D^*(\mathbf{x}) = \frac{1}{2}$

Summarising, we have:

- Optimal Discriminator: $D^*(\mathbf{x}) = \frac{P_{\texttt{real}}(\mathbf{x})}{P_{\texttt{real}}(\mathbf{x}) + P_{\texttt{g}}(\mathbf{x})}$
- $\min_G V(G, D^*) = 2Div_{\texttt{JS}}(P_{\texttt{real}} || P_{\texttt{g}}) - 2\log 2$

Thus,

- $\min_G V(G, D^*)$ is obtained when $P_{\texttt{real}} = P_{\texttt{g}}$,
- $D^*(\mathbf{x}) = \frac{1}{2}$
- and $\min_G \max_D V(G, D) = -2\log 2$

Proof with more details

1. https://srome.github.io/An-Annotated-Proof-of-Generative-Adversarial-Networks-with-Implementation-Notes/
2. Generative Adversarial Nets: (`https://arxiv.org/pdf/1406.2661.pdf`)

Training GANs: Training the Discriminator:

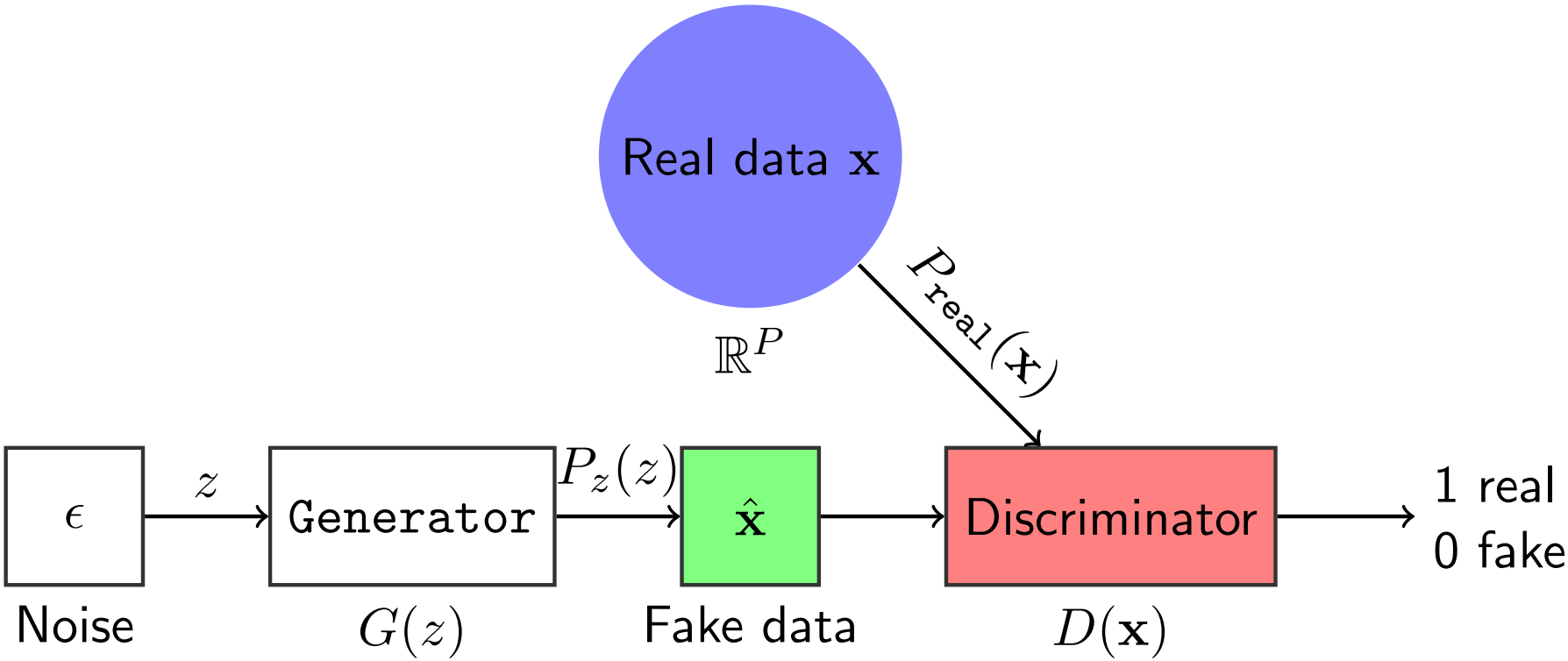


Figure: Only Discriminator should be updated

Training GANs: Training the Generator

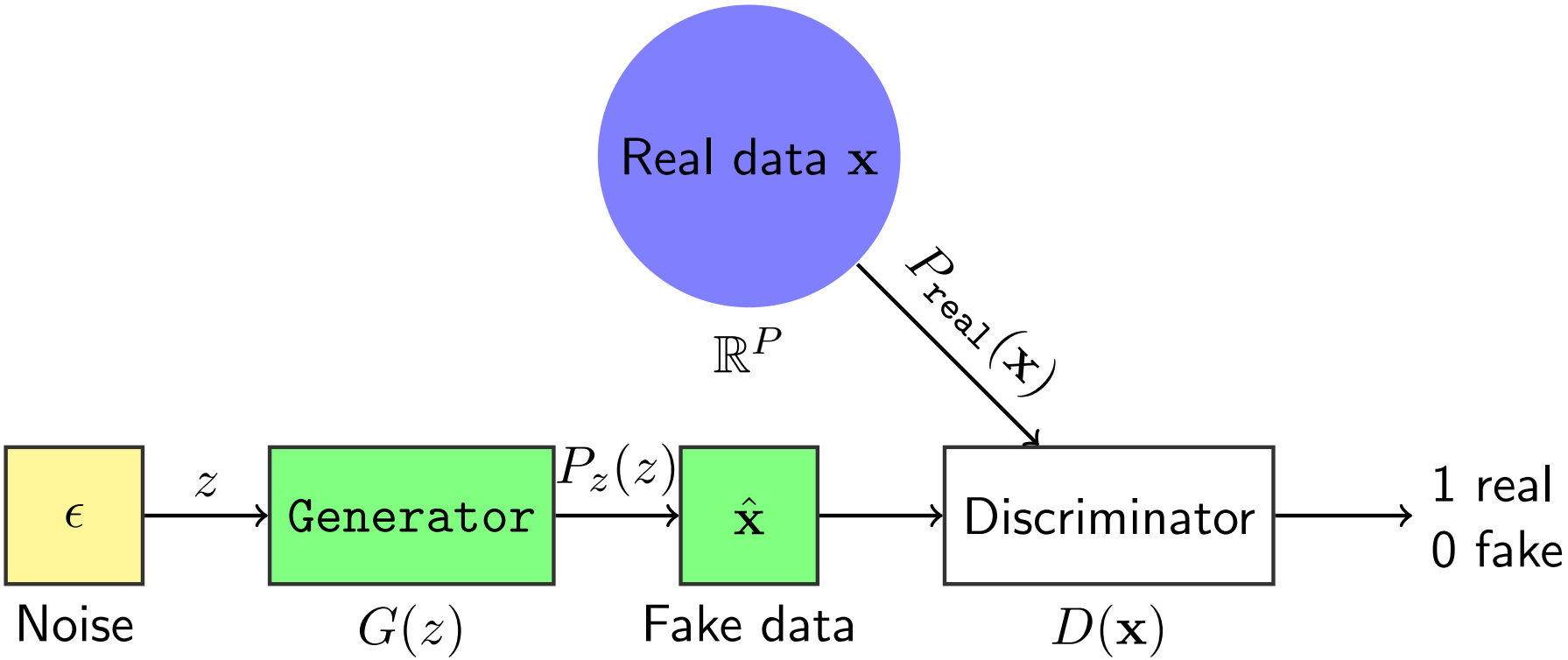


Figure: Discriminator should be set to "non-trainable" weights

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."

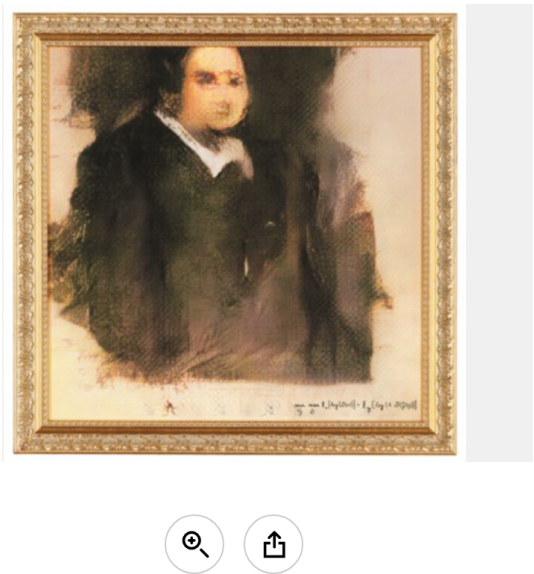
High-resolution GANs

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

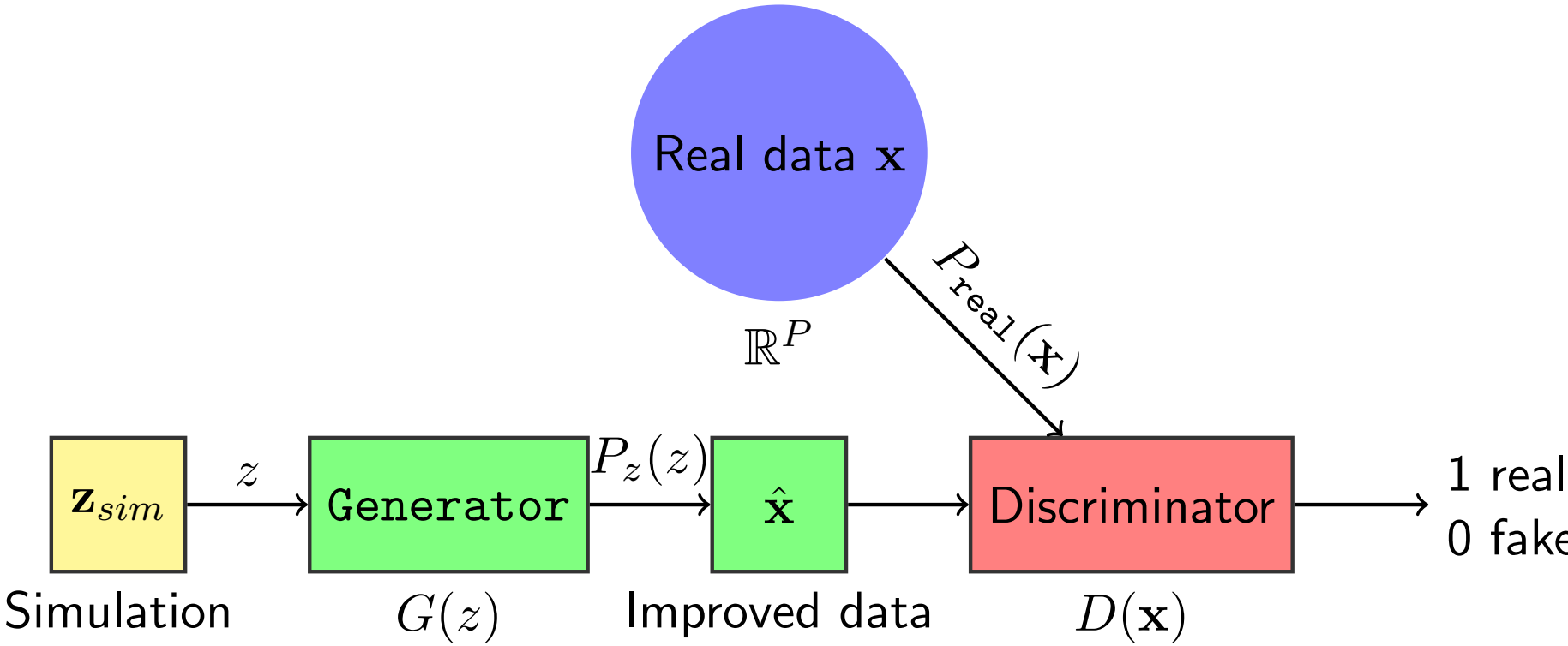


Figure: In this case, `Generator` is called `Refiner`

Application: Refine simulations

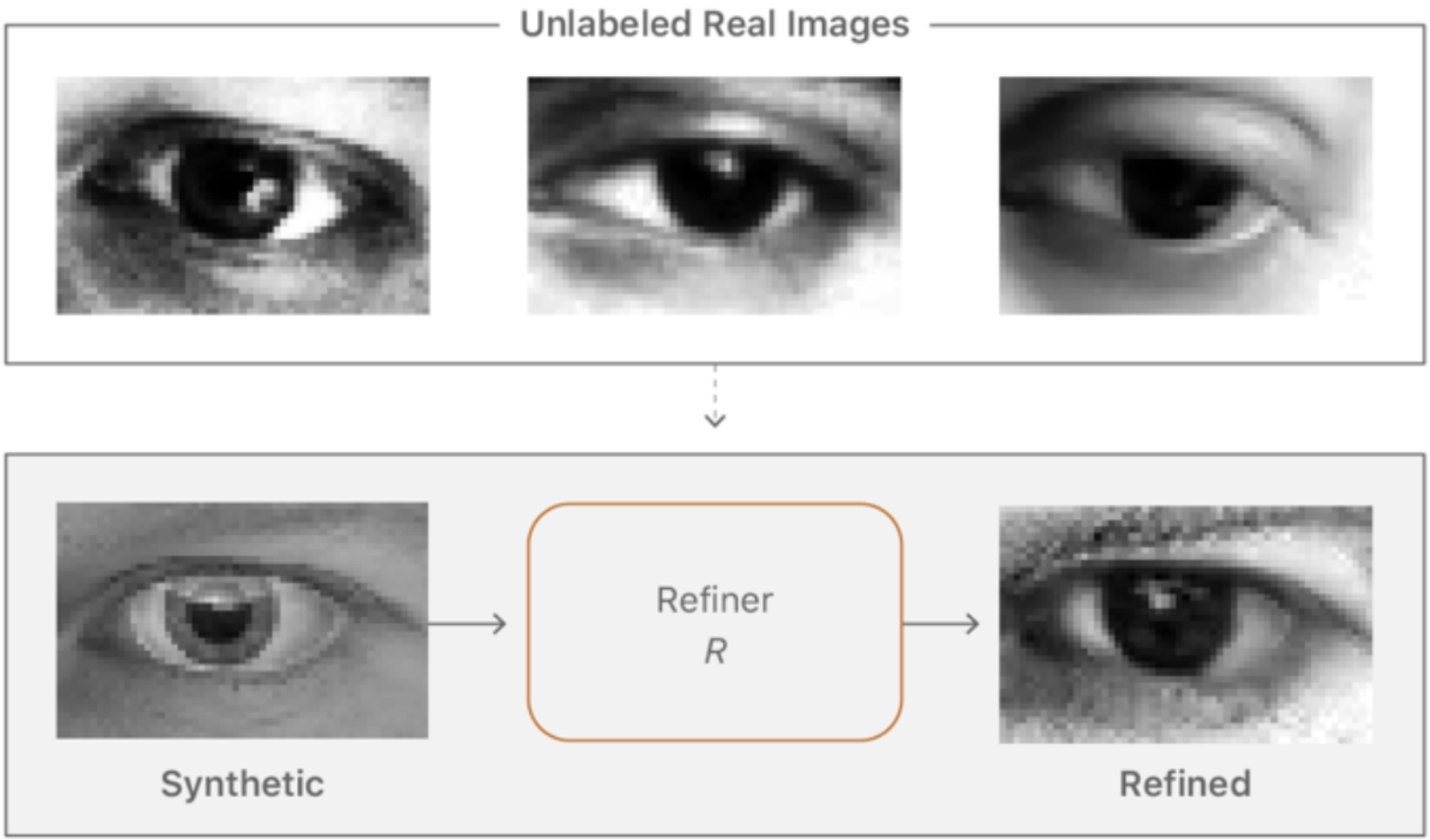
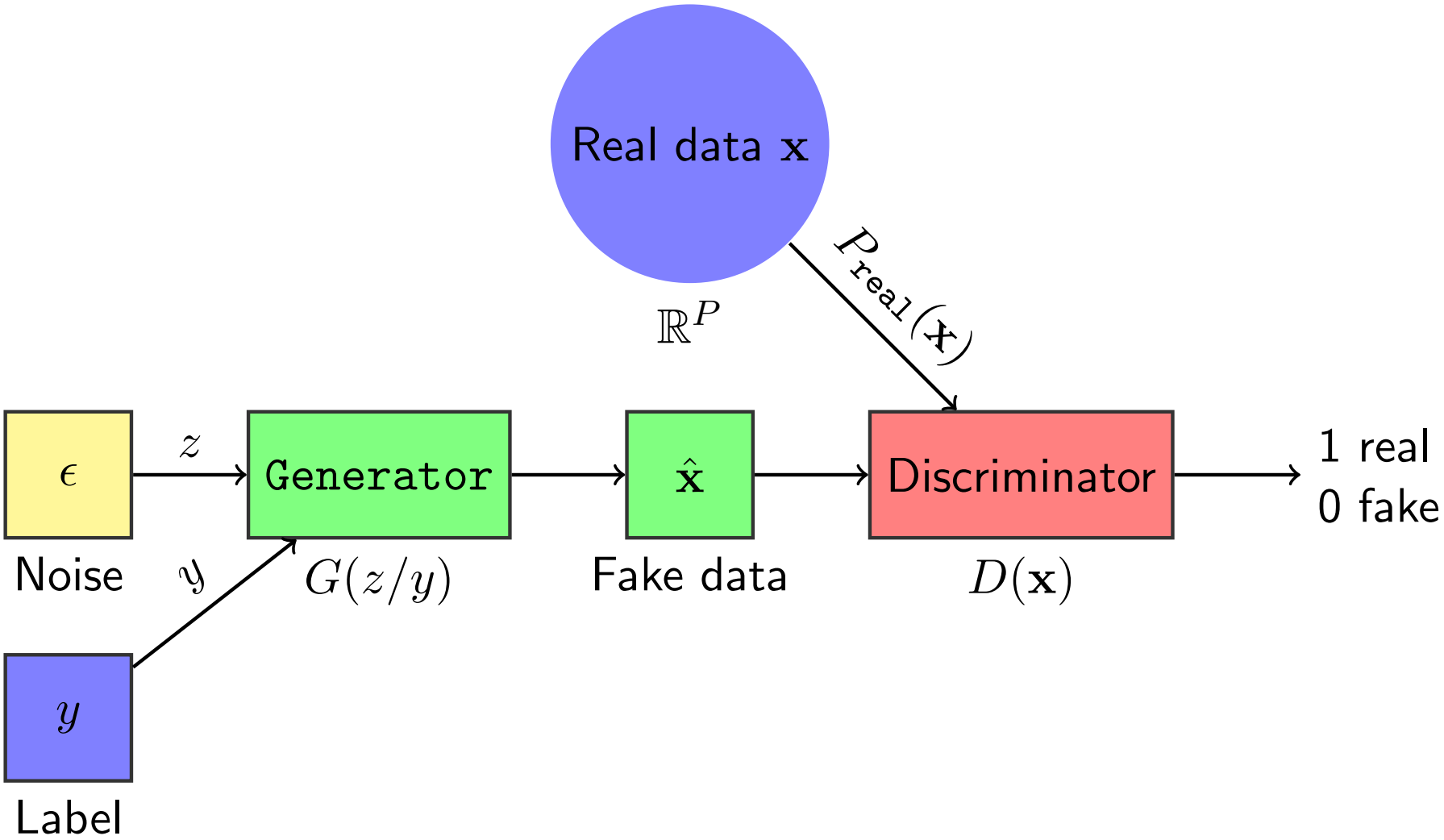


Figure: Learning from Simulated and Unsupervised Images through Adversarial Training [Shrivastava et al., 2017]

Conditional Generative Adversarial Networks (cGANs)



cGANs [Antipov et al., 2017]

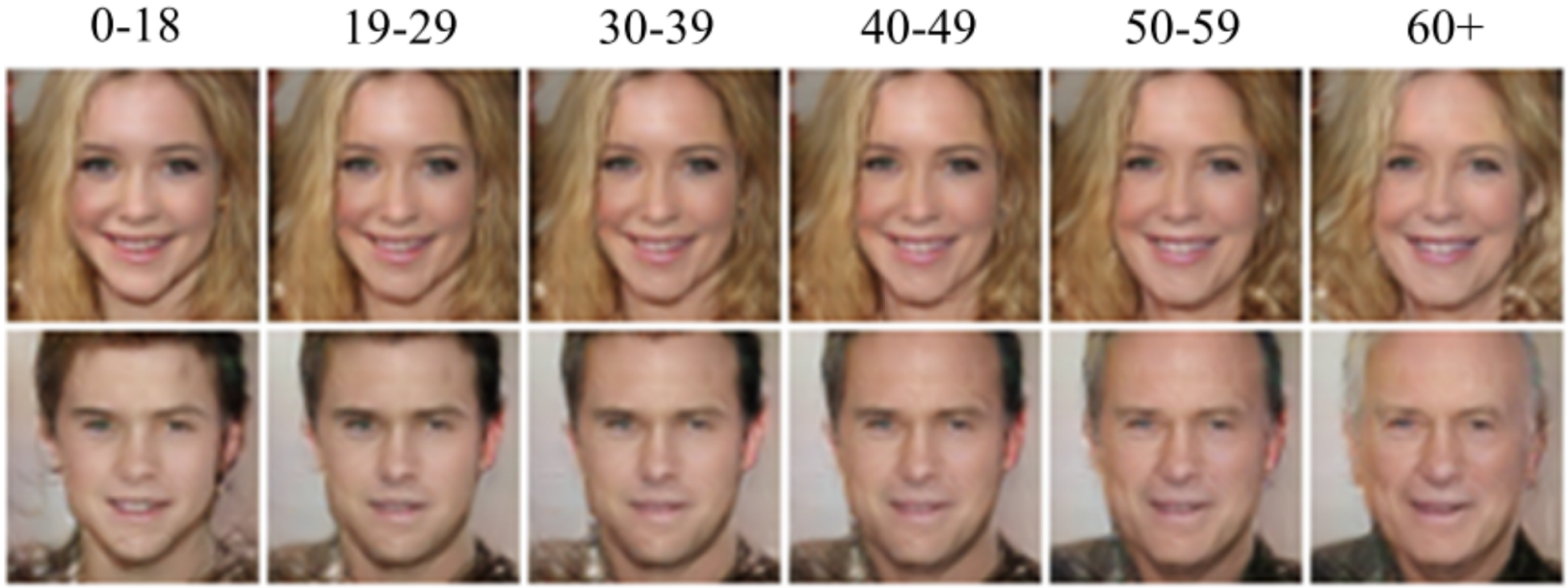


Figure: Face aging with conditional generative adversarial networks

GANs for Pix2Pix

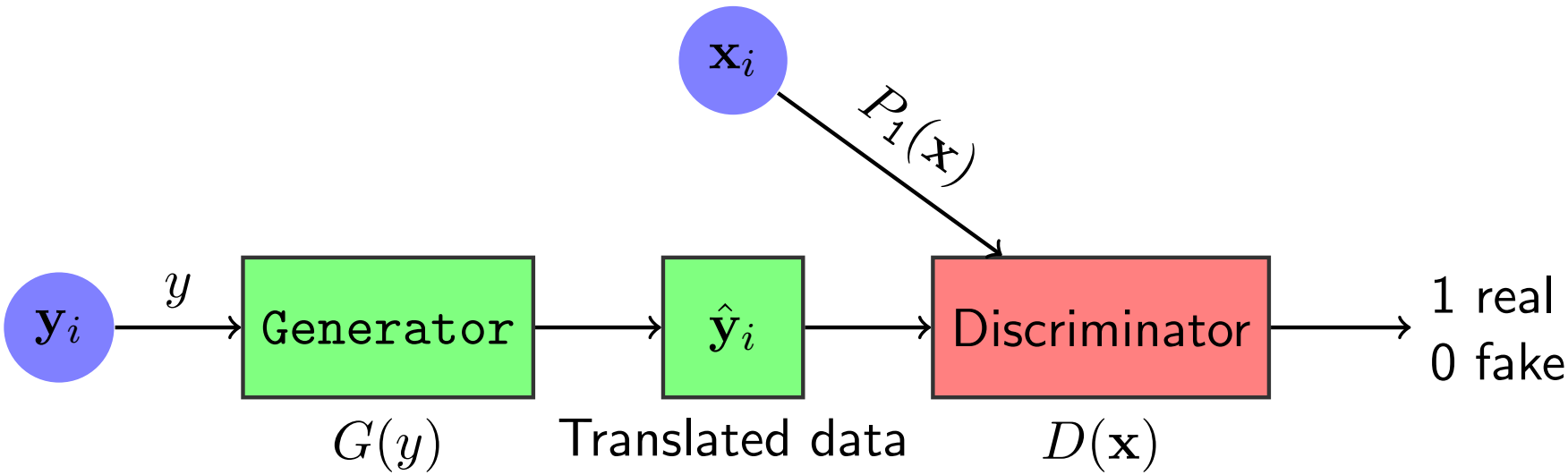


Figure: Training data are pairs of image: $(\mathbf{x}_i, \mathbf{y}_i)$ for $i \in 1, \ldots, N$. In this case, `Generator` is called `Translator`

Note: the discriminator takes as input both images, $(\mathbf{x}_i, \mathbf{y}_i)$.

Example: Colab

1. Simple GANS
   `https://colab.research.google.com/drive/1Zft81l92IJreLiLHBwe5n-HZPGSRQFif?usp=sharing`

Pix2Pix transformation

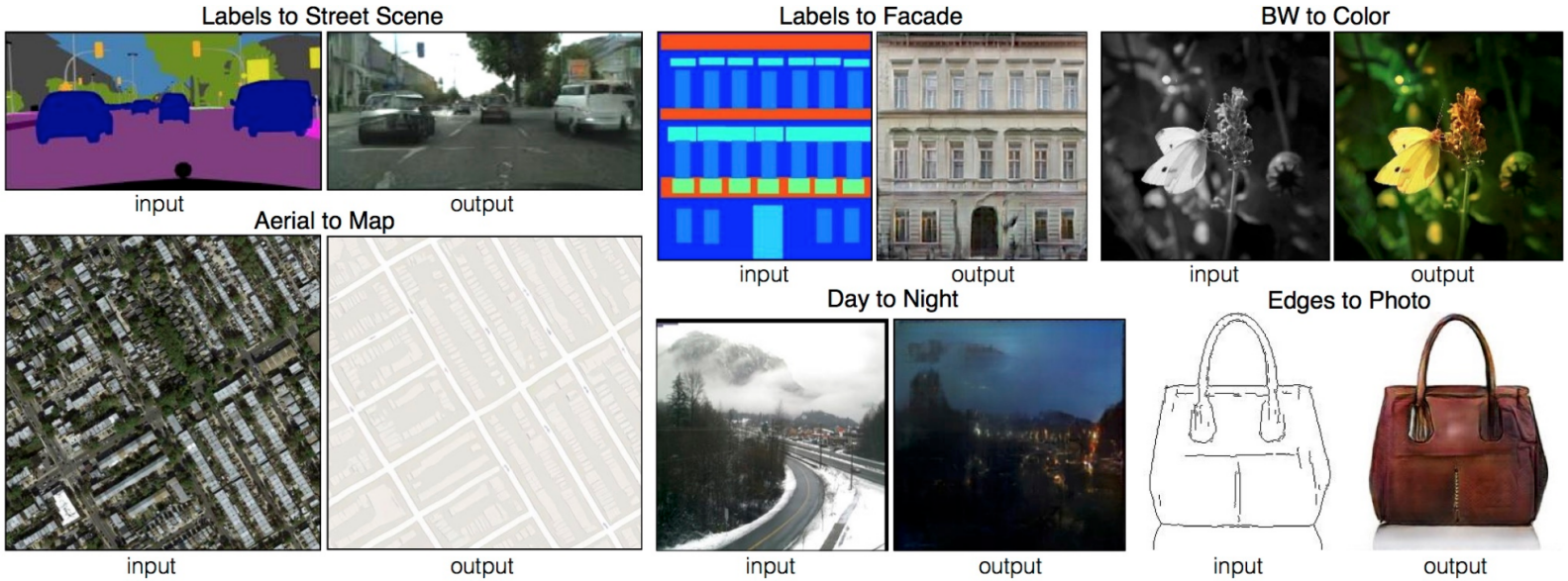


Figure: Example of Pix2Pix, a.k.a. image-to-image translation, [Isola et al., 2017]

Semantic Image Synthesis with Spatially-Adaptive Normalization [Park et al., 2019]

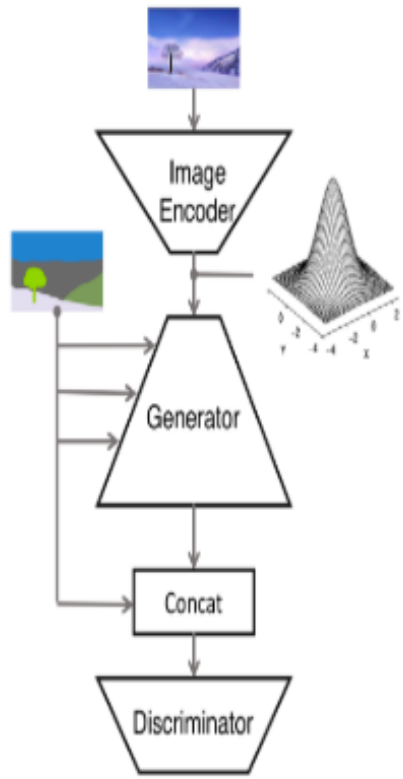


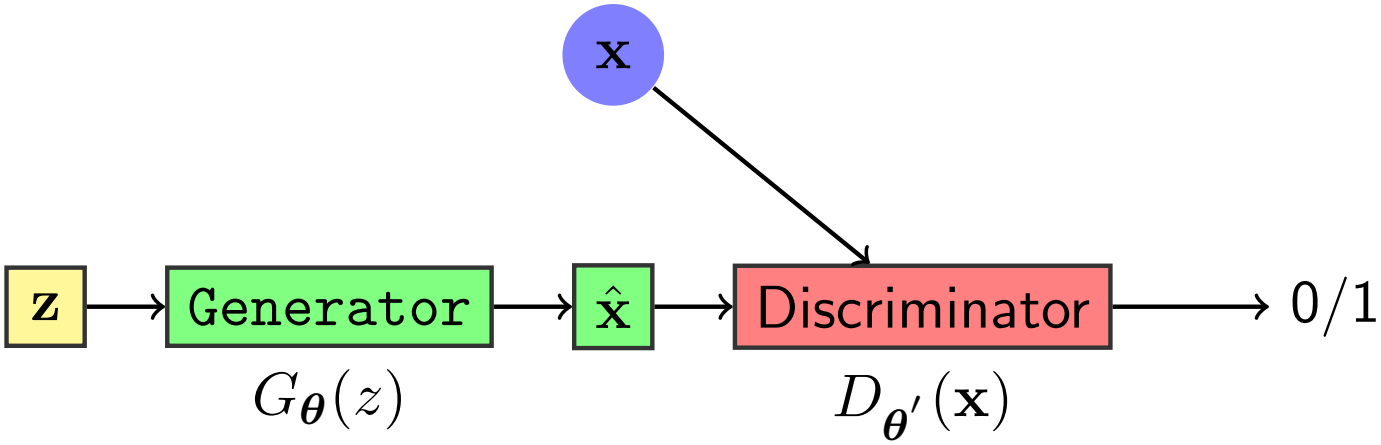
Figure: Use SPADE new module in the generator

`http://gaugan.org/gaugan2/`

Conclusion on GANs

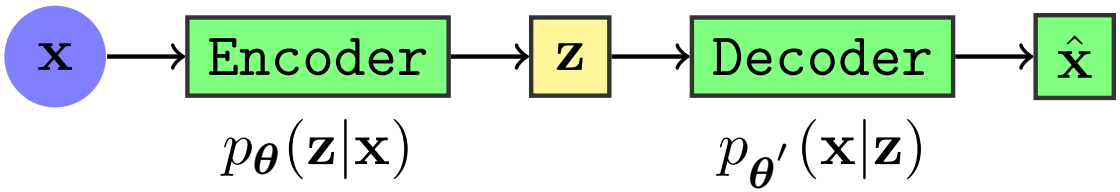
- Interesting for generation from data.
- Conditioning can help to include priori information.
- Generated images (as well as other types of data) are already very realistic

Generative Adversarial Networks (GANs)



- Two steps training: Discriminator and Generator.
- In "testing" only the Generator is used.
- Pretty fast in "testing"
- Hard to optimise.

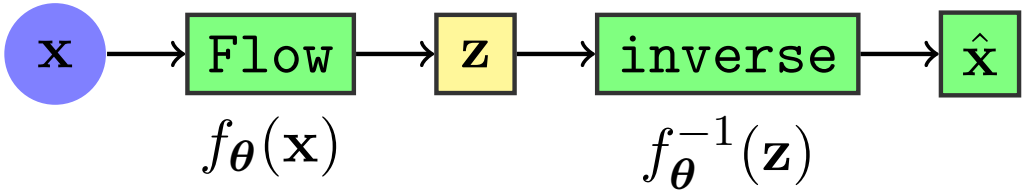
Variational Autoencoders



- One step training: Autoencoder and Decoder.
- In "testing" only the Decoder is used and the generation is done by "walking" in the feature space.
- Pretty fast in "testing"
- Easy to use in simple case. Not necessary a good idea in hard problems.

Contents

Flow-based models



- Flow models restrict the function to be a chain of invertible functions, called a *flow*, therefore the whole function is *invertible*.
- Given $p_z(\mathbf{z})$ where $\mathbf{x} = f(\mathbf{z})$ and $\mathbf{z} = f^{-1}(\mathbf{x}))$ we are looking for $p_x(\mathbf{x})$

$$p_x(\mathbf{x}) = p_z(f^{-1}(\mathbf{x})) \left| \det \left(\frac{\partial f^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \right| \tag{13}$$

SIREN

SIREN is a neural network using a periodic activation function, i.e., each **layer** has the following expression:

$$\texttt{SIREN}(\mathbf{x}) = f(\mathbf{x}) = \sin(\mathbf{W}_i \mathbf{x}_i + \mathbf{b}_i) \tag{14}$$

1. Any derivative of a SIREN is itself a SIREN.

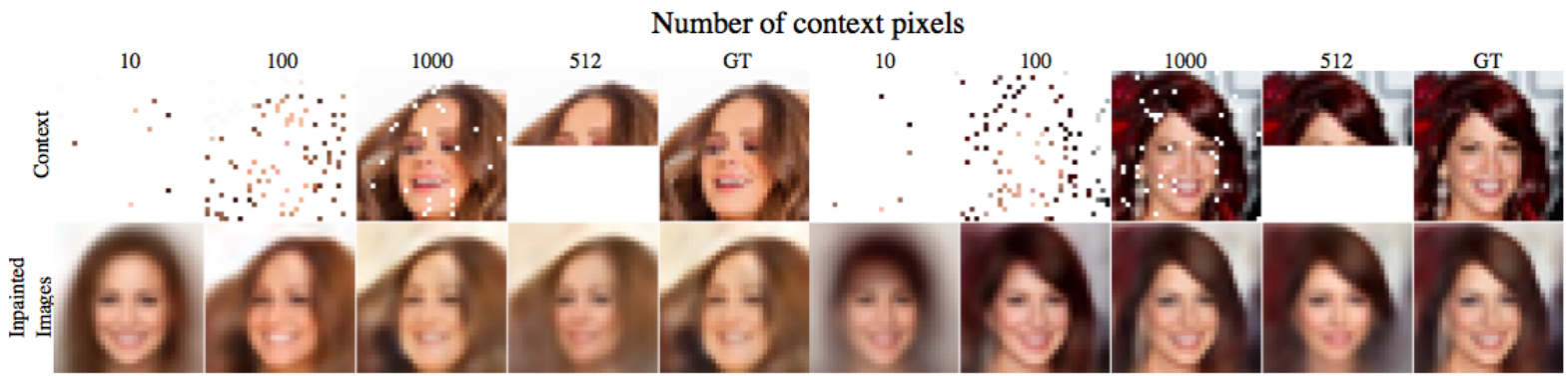
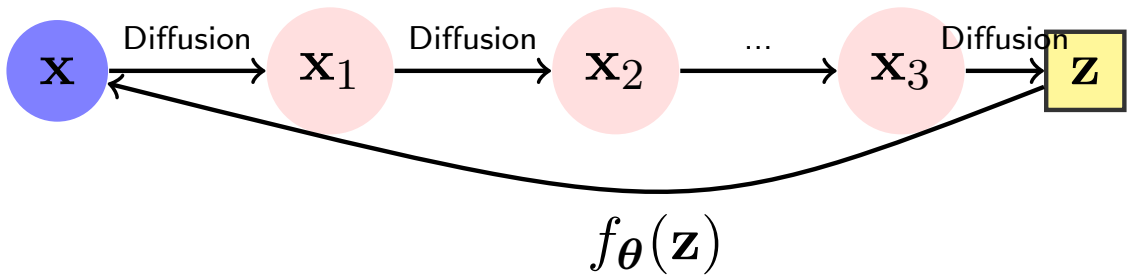


Figure: SIREN [Sitzmann et al., 2020]

Contents

Diffusion models



- The *forward diffusion* process is defined as add a small amount of Gaussian noise to the sample T steps producing a sequence of noisy samples $\mathbf{x}_1, \ldots, \mathbf{x}_T$

Diffusion models

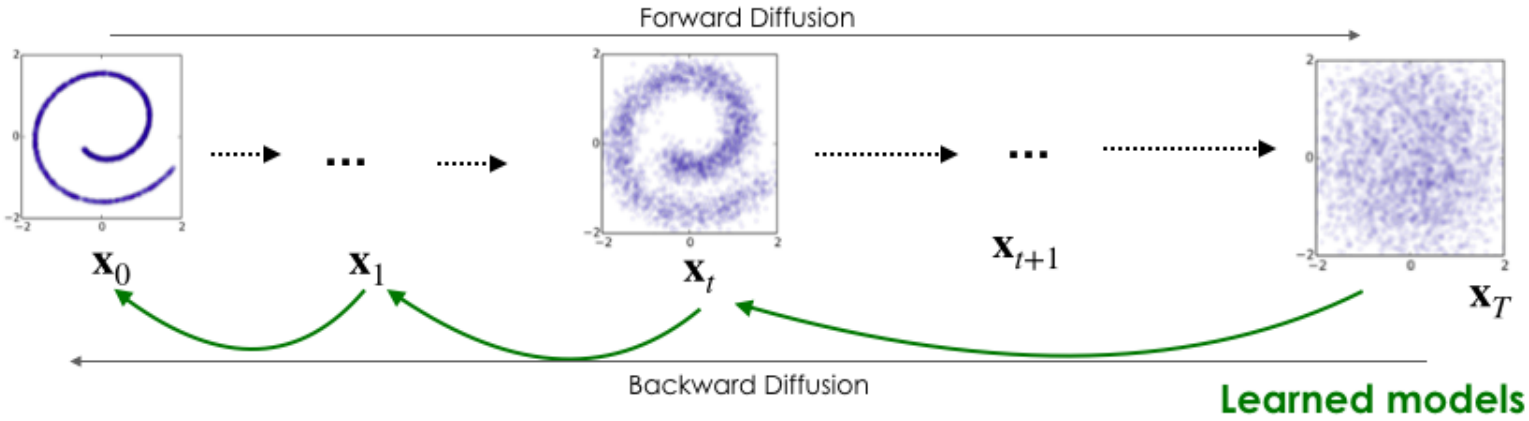


Figure: Diffusion models

Diffusion models

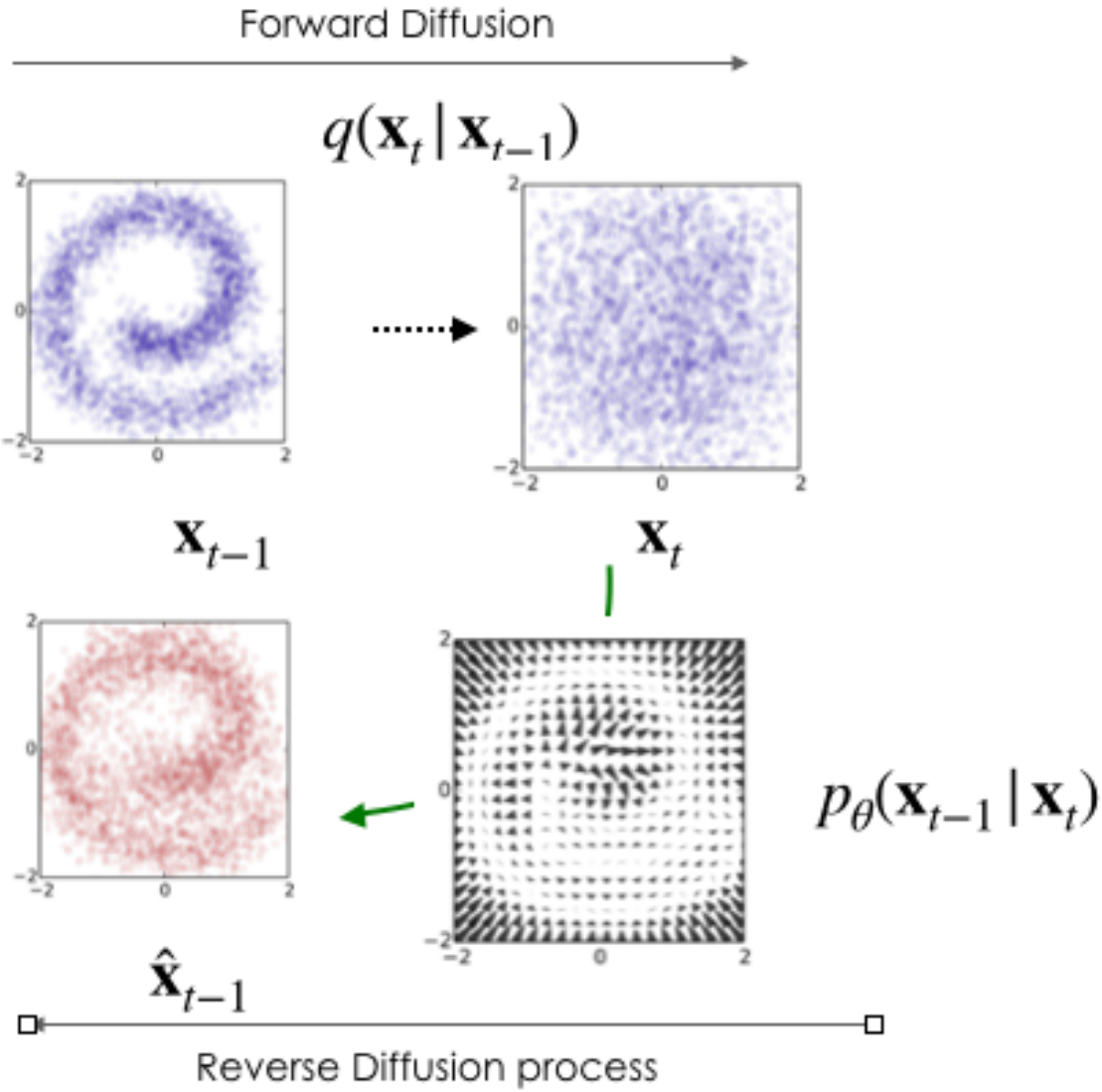


Figure: Diffusion models

Stable Diffusion models

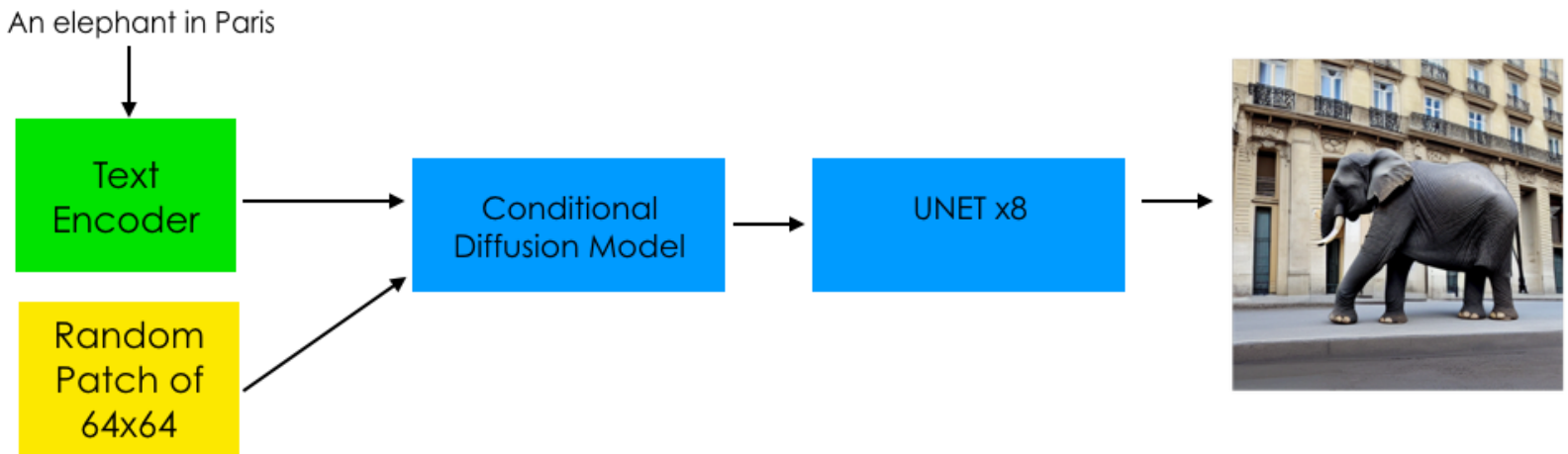


Figure: StableDiffusion

Stable Diffusion models

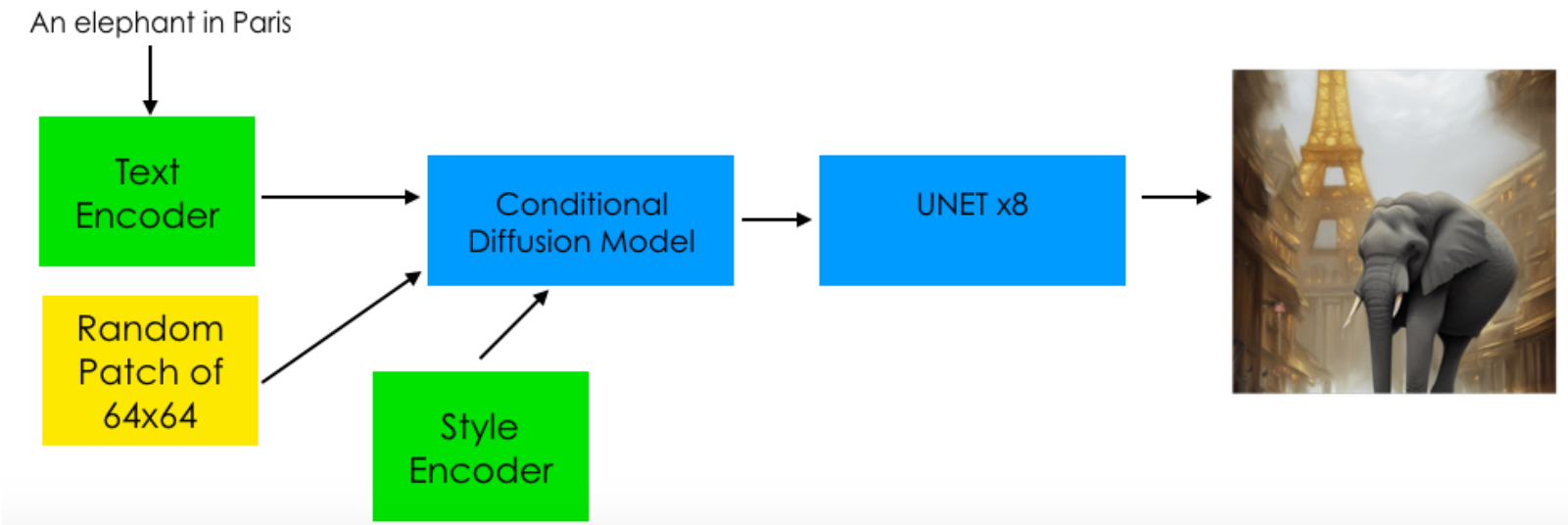


Figure: StableDiffusion

Example: Colab

1. Stable Diffusion Model
   `https://colab.research.google.com/drive/1xmskXN15B3vK_HtwKy_HNFlKgCPgemiT?usp=sharing`

Questions

- The word “adversarial” in the acronym for GANs suggest two-players. What are the two players, and what are their respective goals?
- What is the difference between adversarial attacks and adversarial networks?
- If the discriminator is able to detect correctly fake images means that GANs converges: True/False
- Suppose you have a code to generate ”city maps” and also a base of real city maps. How can you try to improve the quality of your generator by considering the real images?

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.

[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.

References II

[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*.

[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*.

[Park et al., 2019] Park, T., Liu, M.-Y., Wang, T.-C., and Zhu, J.-Y. (2019). Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2337–2346.

References III

[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*.

[Sitzmann et al., 2020] Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G. (2020). Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473.

[Su et al., 2019] Su, J., Vargas, D. V., and Sakurai, K. (2019). One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841.

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.