Deep Learning Fundamentals

Mihai DOGARIU

www.mdogariu.aimultimedialab.ro

Summary

M5. Supervised Deep Learning (general info)

M6. Classification

M7. Unsupervised Deep Learning (general info)

M8. Autoencoders (AEs)

M9. Variational Autoencoders (VAEs)

M10. Generative Adversarial Networks (GANs)

M5. Supervised Deep Learning

Supervised Deep Learning

Machine learning = we say that a system „learns” from experience E regarding a set of tasks T and a performance measure P, if the performance in solving the tasks T, measured by P, grows with the experience E.

Dataset = a group of elements with common properties. It represents the „experience” that an algorithm makes use of when learning a certain task.

$$D = \{((x_i, y_i)|T), 1 \leq i \leq M\}$$

input output task dimension

Supervised Deep Learning

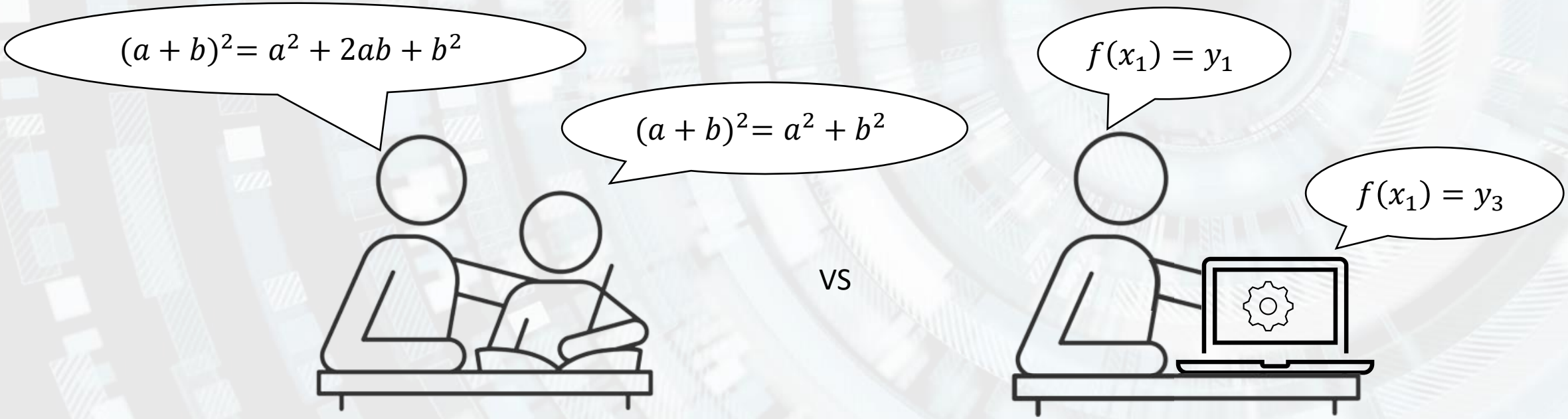
$$D = \{((x_i, y_i)|T), 1 \leq i \leq M\} = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_M, y_M)\}$$

$$\left.\begin{aligned} f(x_1) &= y_1 \\ f(x_2) &= y_2 \\ f(x_3) &= y_3 \\ &\dots \\ f(x_M) &= y_M \end{aligned}\right\} \; f = ?$$

- Each (x_M, y_M) pair is called a training sample;
- x_M = input vector;
- y_M = real output/label.

Sursă imagine: iStock

Supervised Deep Learning



Supervised Deep Learning

Supervised Learning = machine learning paradigm in which training data is labeled. Each training sample is composed of a descriptor and a label. The goal of supervised learning is to learn the association between the input features and their corresponding labels.

Supervised Deep Learning = supervised machine learning paradigm applied to deep neural networks (deep = several (>3, >7, >30, >50) layers).

Supervised Deep Learning

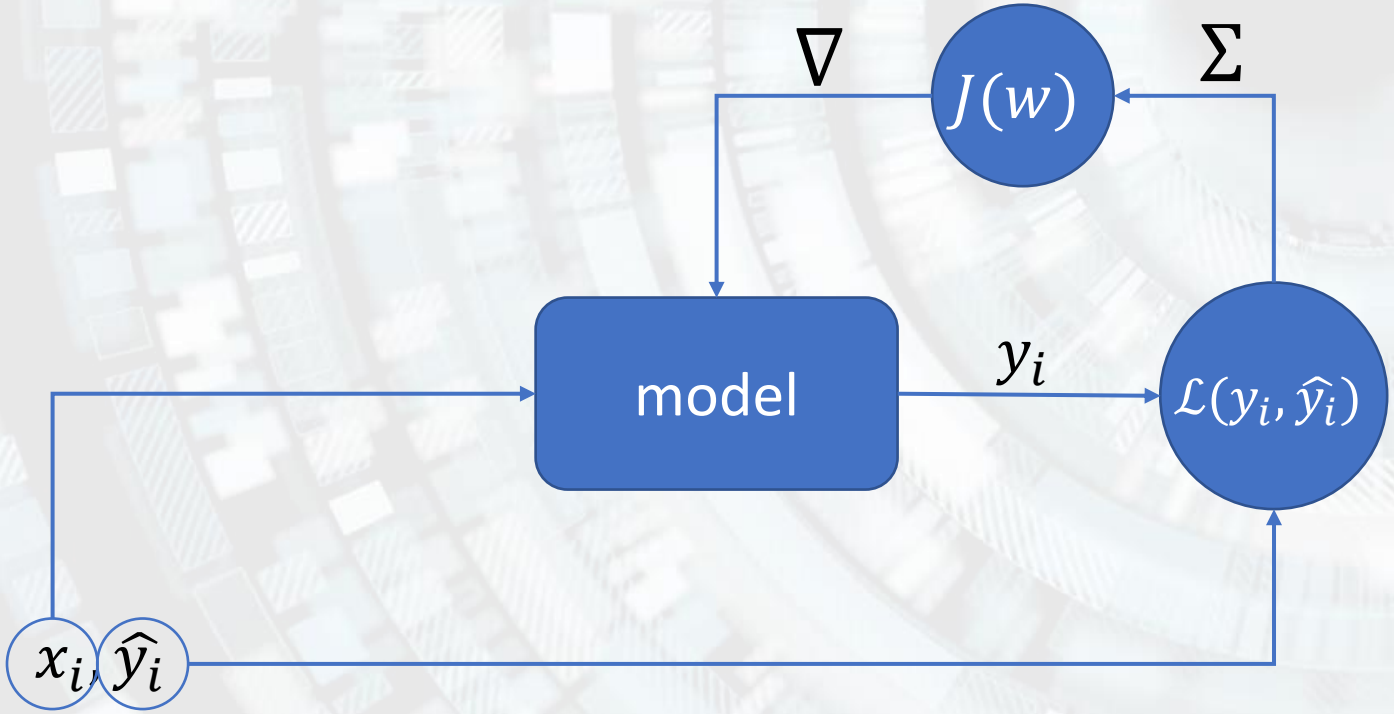
- All training samples contain a label of their own;

- 2 main subcategories:
  1. Regression – we would like to predict continuous values, adapted to the model that describes the dataset.
  2. Classification – we would like to predict a discrete value, representing the class to which an input sample belongs.

- Definition collision:
  - A classifier can predict a continuous value under the form of a probability distribution.
  - A regressor cand predict a discrete valuea under the form of an integer quantity.

Let's test it out – unit #6

Supervised Deep Learning



M6. Classification

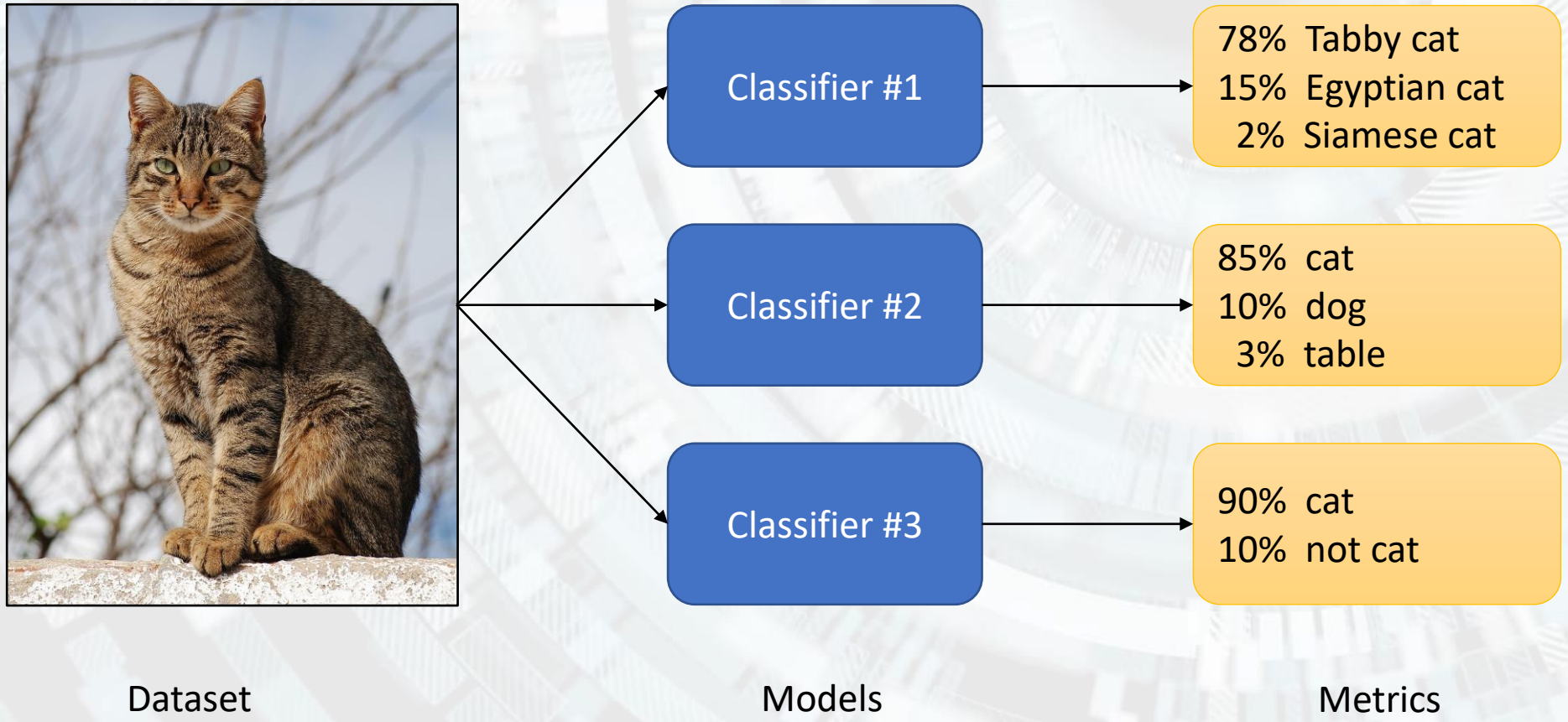
Classification

Classification = the task of assigning a label to a sample.

Characteristics:

- The content of the sample is treated as a whole ⇔ the label describes the entire input, not just a part of it.
- Any sample belongs to a single class.
- The output of a classifier is usually a probability, not a categorical decision.
- Usually, we classify samples where the object of interest takes up most, if not all, of the sample.
- Strongly depends on the qualitative and quantitative aspects of the training dataset.
- Multi-class classifiers usually have the last layer fully connected and a softmax activation.

Classification



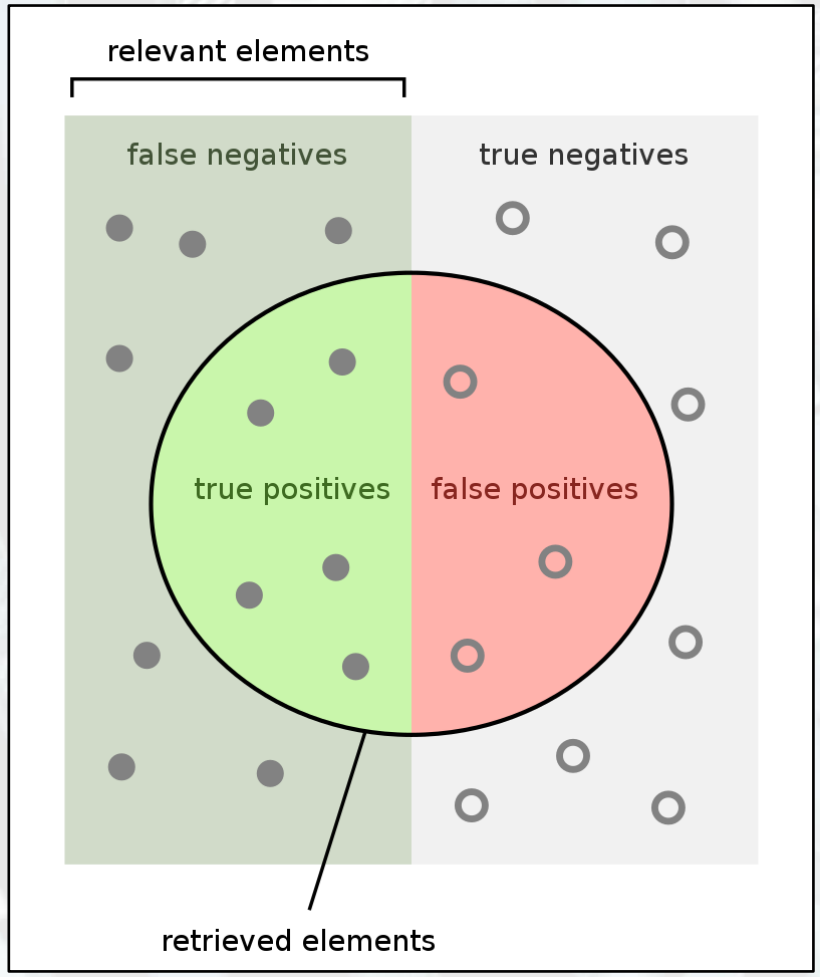
Classification – metrics

Metric = quantitative method that is used to measure the performance of a system. It offers sortable numerical values such that we can quantify the progress made by a trainable model.

- It is calculated at the end of an epoch, on the entire dataset (train, val, test).
- It depends on the workload – different metrics for different types of systems.
- It is usually not differentiable, so it cannot be used to drive the learning process.
- In some cases, it can be synonymous with the cost function (e.g. MAE, MSE).
- It must be interpreted in context. Usually, the values/ranges of values that represent the desired situation are mentioned.

Image: Wikipedia

Classification – metrics



How many retrieved items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{retrieved elements}}$$

How many relevant items are retrieved?

$$\text{Recall} = \frac{\text{true positives}}{\text{relevant elements}}$$

*Only for binary classification!

Classification – metrics

Confusion matrix:

Real \ Predicted	Positive	Negative
Positive	True positive (tp)	False negative (fn)
Negative	False positive (fp)	True negative (tn)

$$precision(P) = \frac{tp}{tp + fp}$$

$$recall(R) = \frac{tp}{tp + fn} = rata\ TP\ (TPR)$$

$$TN\ rate\ (TNR) = \frac{tn}{tn + fp}$$

$$FP\ rate(FPR) = \frac{fp}{tn + fp}$$

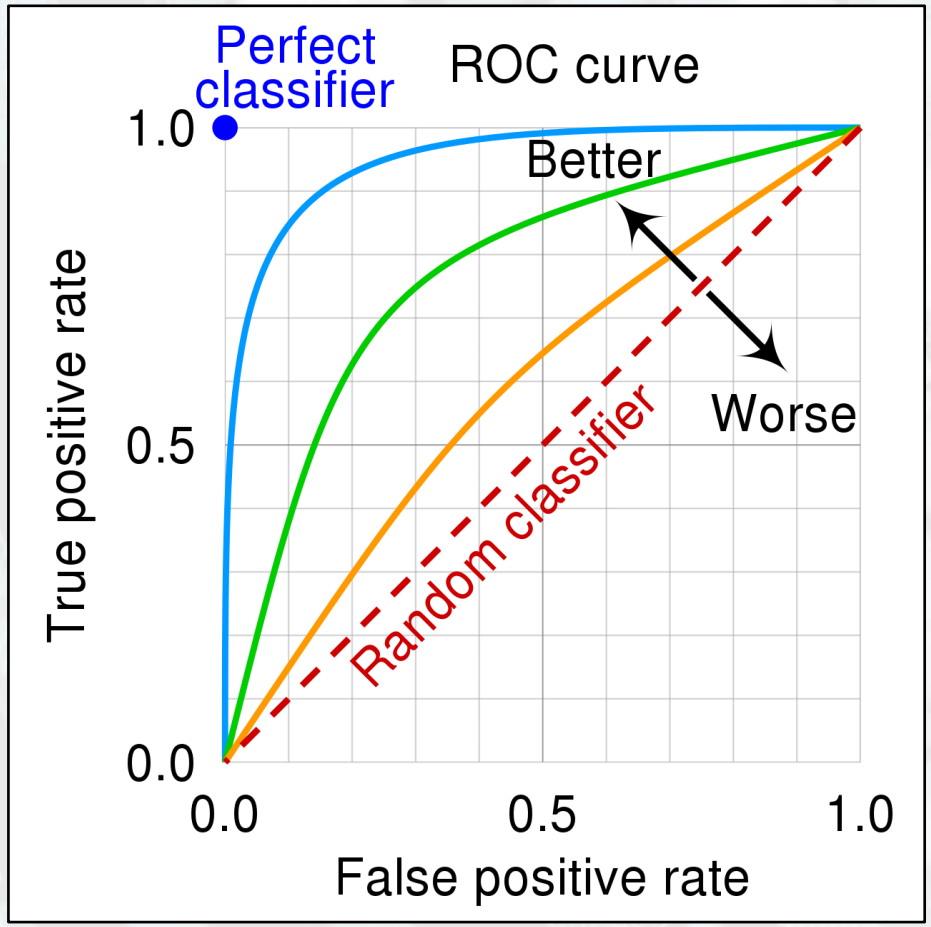
$$accuracy\ (ACC) = \frac{tp + tn}{tp + tn + fp + fn}$$

$$F - score\ (F) = 2\frac{PR}{P + R}$$

*Only for binary classification!

Image: Wikipedia

Classification – metrics



ROC = Receiver Operating Characteristic
AUC = Area Under the (ROC) Curve
AUC=1 => perfect classifier
AUC=0.5 => random classifier
AUC=0 => anti-perfect classifier

Classification – metrics

Accuracy for imbalanced datasets

- For a binary classifier, the accuracy is defined as:

$$ACC = \frac{tp + tn}{tp + tn + fp + fn}$$

- Scenario:
  - Consider a dataset with 95 negative examples and 5 positive examples.
  - A classifier which predicts the negative class in 100% of the cases will score a 95% accuracy, which is deceiving.
  - Solution: use balanced accuracy.

Classification – metrics

Accuracy for imbalanced datasets

- Balanced accuracy is defined as:

$$ACC = \frac{\text{TPR} + \text{TNR}}{2} = \frac{\frac{tp}{tp + fn} + \frac{tn}{tn + fp}}{2}$$

- Scenario:
  - Consider a dataset with 95 negative examples and 5 positive examples.
  - A classifier which predicts the negative class in 100% of the cases

		Predicted: Positive	Predicted: Negativ
Real	Positive	tp = 0	fn = 5
	Negative	fp = 0	tn = 95

$$ACC = \frac{\text{TPR} + \text{TNR}}{2} = \frac{\frac{0}{0 + 5} + \frac{95}{95 + 0}}{2} = 0.5$$

Classification – metrics

Accuracy for multi-class classification:

$$ACC = \frac{\#correct\ classifications}{\#total\ classifications}$$

- E.g.: from 100 analyzed examples, 83 were correctly classified => 83% accuracy.

Top-k accuracy

- For an example from the database, the relative probability of belonging to each class is calculated.
- The output probabilities are ordered.
- If n among the first k best rated classes is also the real class => correct classification.

Classification – metrics

Classifier

35%	**cat**
3%	table
1%	velociraptor
12%	airplane
6%	truck
40%	dog
2%	baseball
1%	mouse

40%	dog	top-1: miss
35%	**cat**	top-2: hit
12%	airplane	top-3: hit
6%	truck	
3%	table	top-5: hit
2%	baseball	
1%	velociraptor	
1%	mouse	

Let's test it out – unit #7

M7. Unsupervised Deep Learning

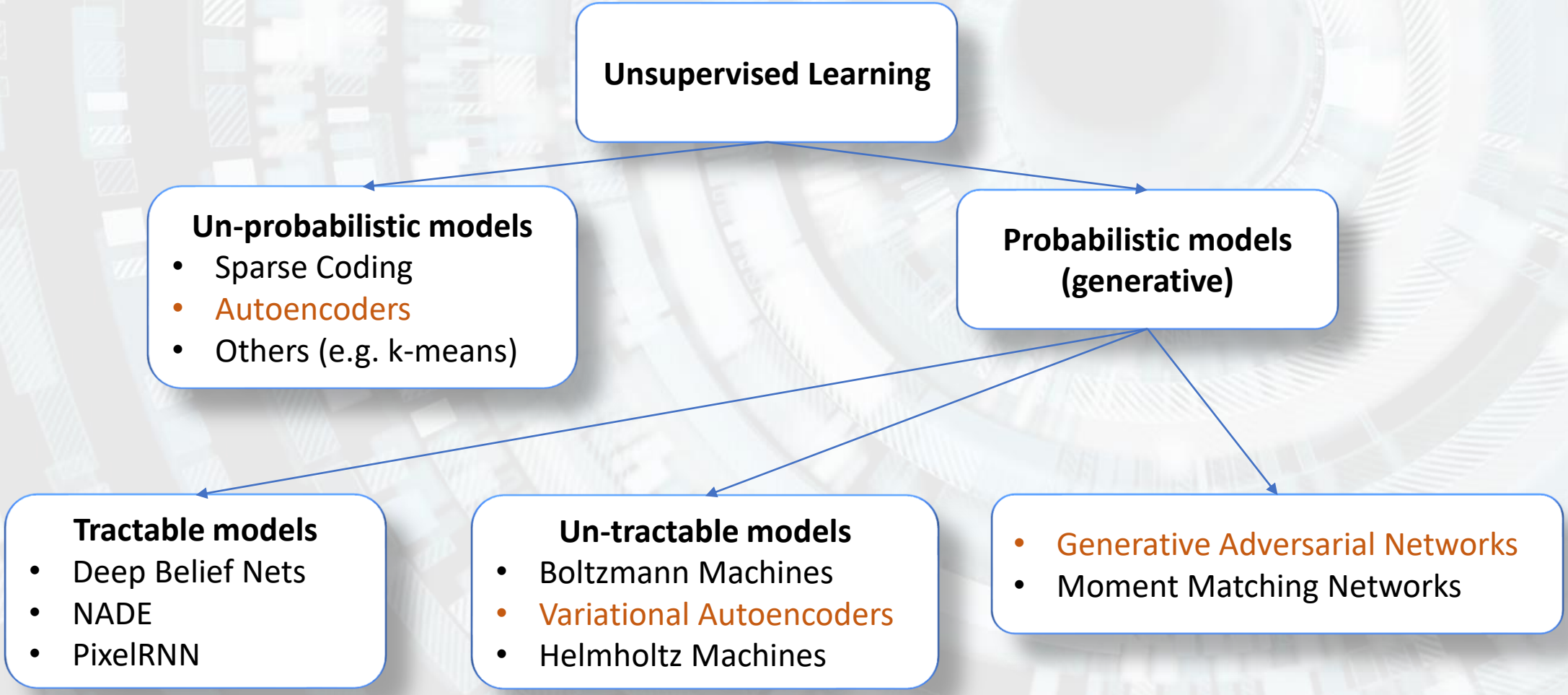
Unsupervised Deep Learning

Supervised Learning = machine learning paradigm in which training data is labeled. Each training sample is composed of a descriptor and a label. The goal of supervised learning is to learn the association between the input features and their corresponding labels.

Unsupervised Learning = machine learning paradigm in which training data is unlabeled. The goal of unsupervised learning is to learn the inherent mode of the training samples through imitation. This is done through self-organization, which captures information relative to the behavior/probability distribution of the training data.

Image: R. Salakhutdinov

Unsupervised Deep Learning



M8. Autoencoders

Autoencoders

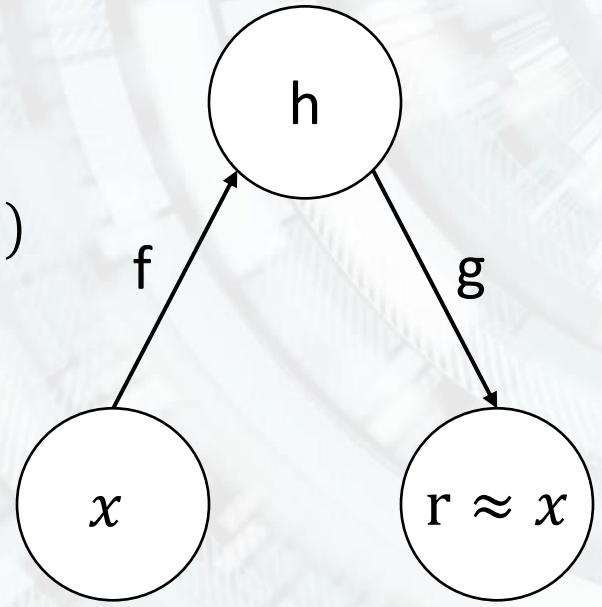
Autoencoder = a neural network trained to reproduce the input at its output.

hidden layer = bottleneck

Deterministic functions

$encoder$: $h = f(x)$

$decoder$: $r = g(h) = g(f(x))$



Stochastic generalization

$encoder$: $p_{encoder}(h|x)$

$decoder$: $p_{decoder}(x|h)$

- f and g can be obtained with the help of a neural network;
- x and r represent informational structures (vectors/tensors);
- h represents a feature vector.

Training is done with a loss

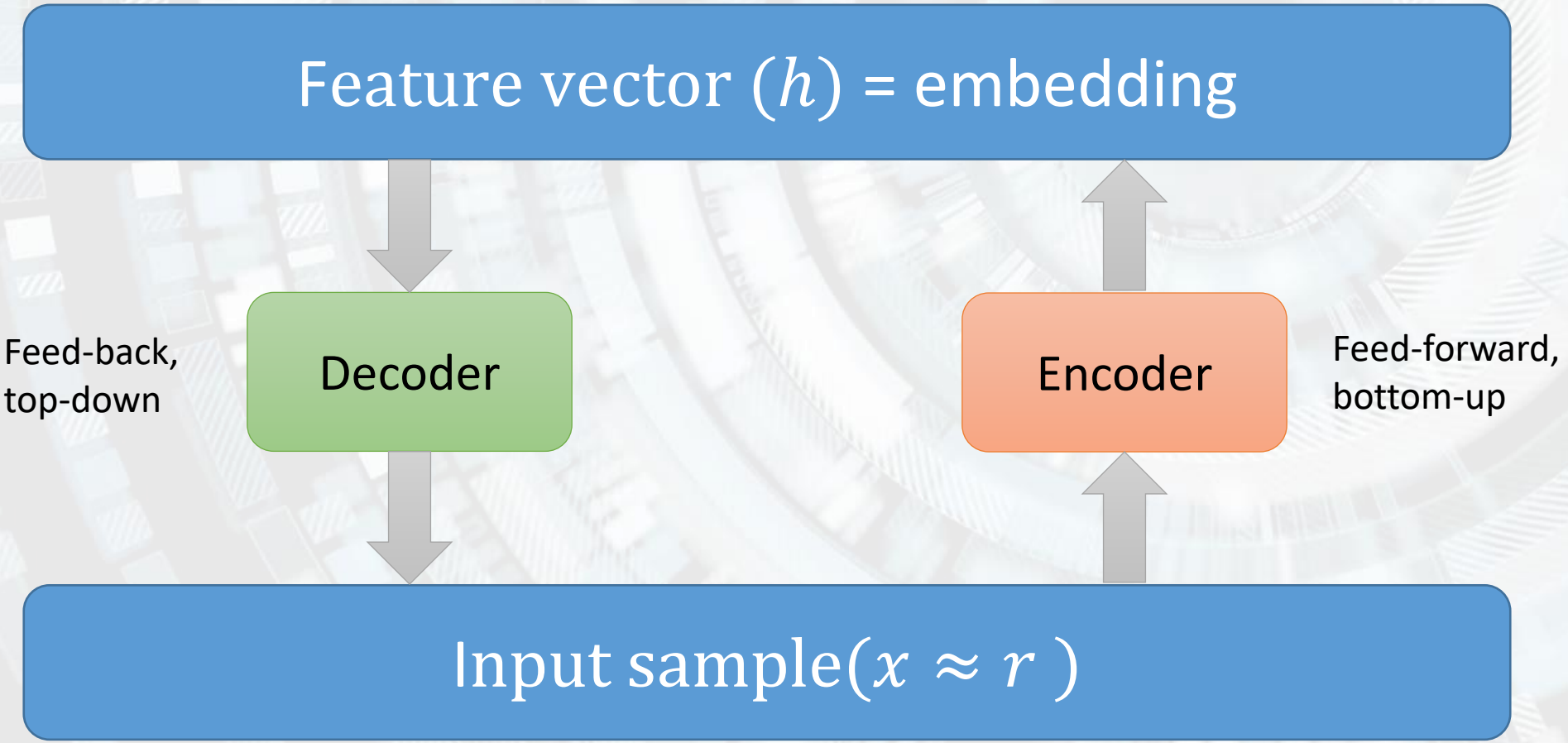
$$L(x, g(f(x)))$$

which penalizes the output $g(f(x))$ when it is different from x; e.g. MSE.

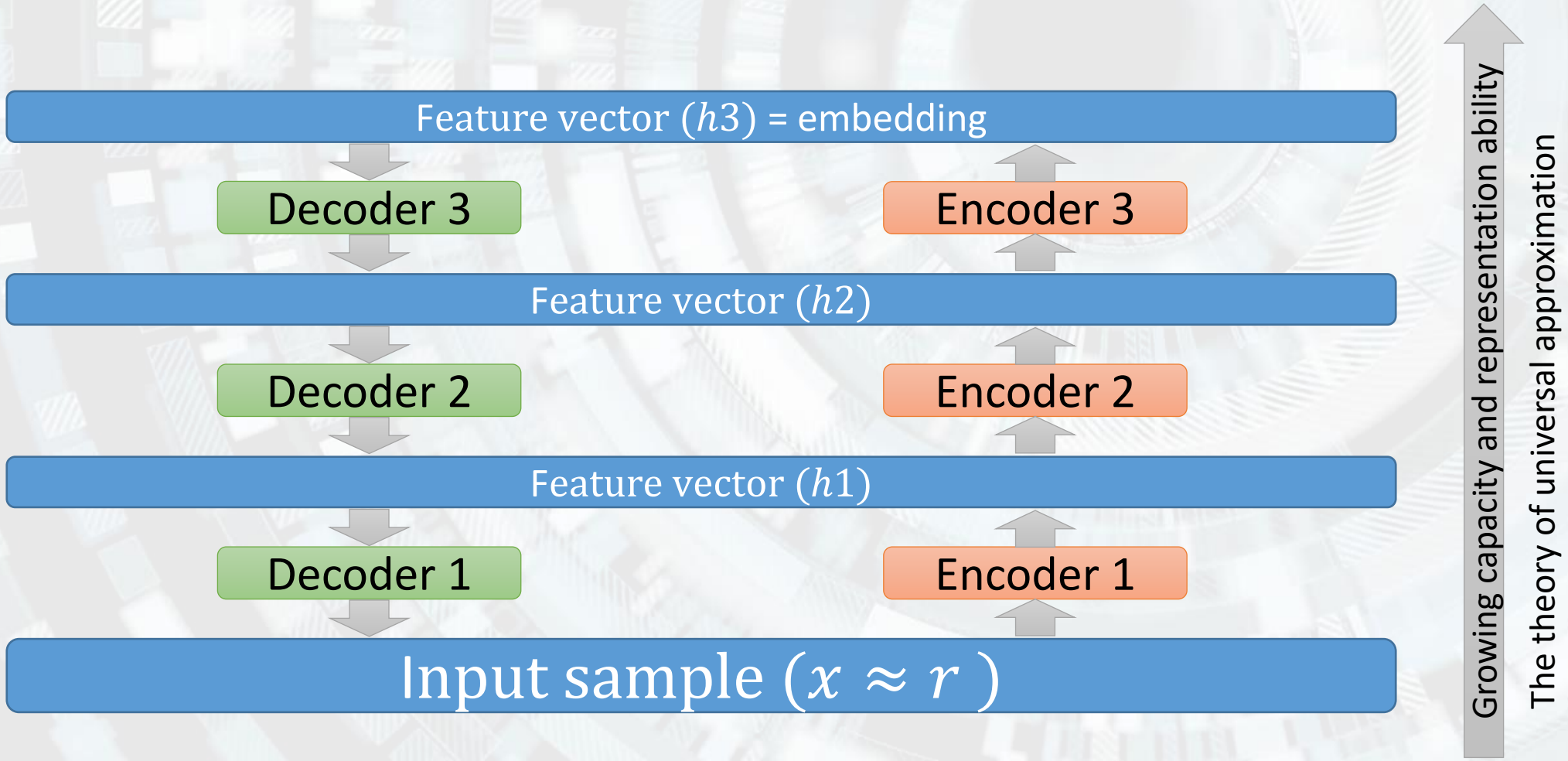
input

output = reconstruction

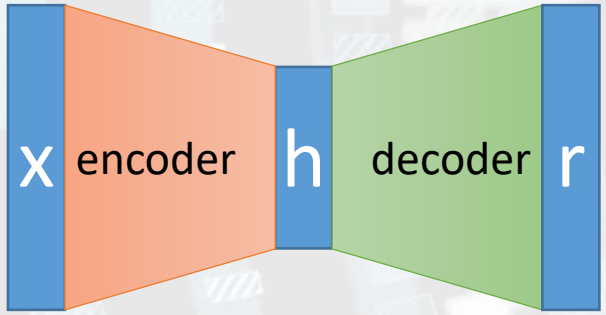
Autoencoders



Autoencoders – stacked



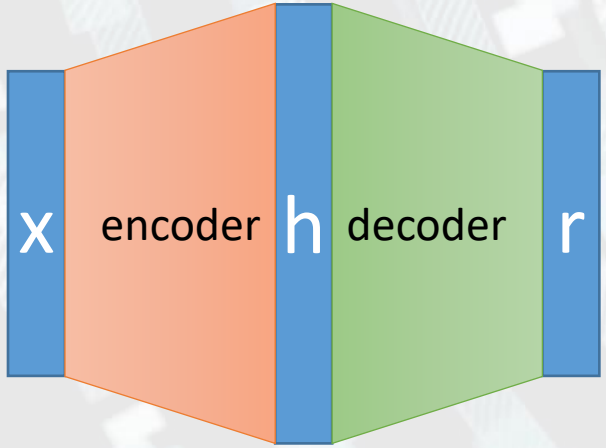
Autoencoders



Undercomplete autoencoder

$\dim(h) < \dim(x)$

The encoder and the decoder learn a contracted representation of the data.

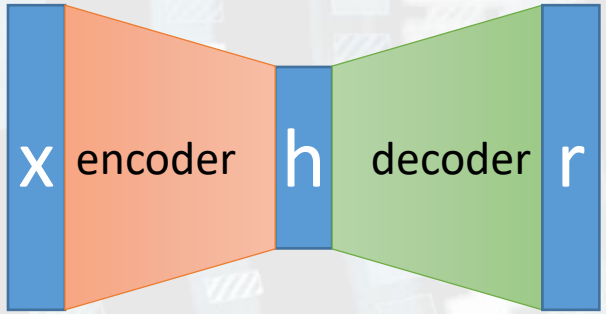


Overcomplete autoencoder

$\dim(h) \geq \dim(x)$

The encoder and the decoder learn a dilated representation of the data.

Autoencoders



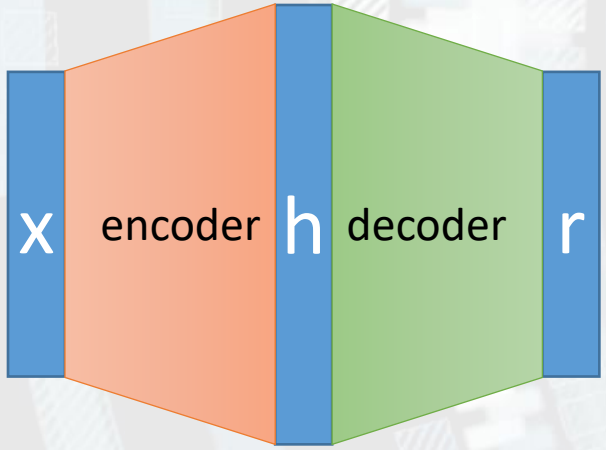
Undercomplete autoencoder

$\dim(h) < \dim(x)$

The encoder and the decoder learn a contracted representation of the data.

- The feature vector is restricted to a smaller size than the input => the autoencoder learns the most representative features;
- If we give too much capacity (number of trainable parameters) to the encoder and decoder, they may end up learning the identity function => the autoencoder becomes useless.

Autoencoders



Overcomplete autoencoder

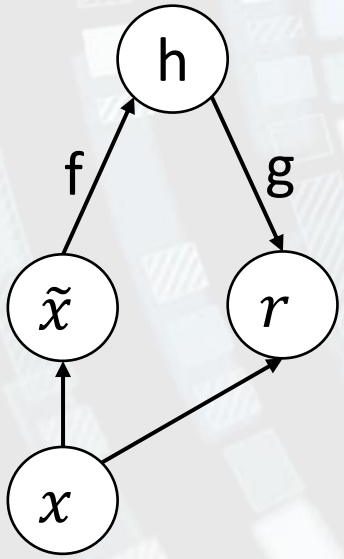
$\dim(h) \geq \dim(x)$

Encoder-ul și decoder-ul învață o reprezentare dilatată a datelor.

- The feature vector has a size at least equal to the size of the data;
- The autoencoder can reproduce the input to the output very simply, by copying the data from one end to the other (linear encoder + linear decoder) – this case is not desired, because the autoencoder does not learn anything;
- The identity function is avoided by regularization – a series of additional constraints that prevent the input from being copied to the output.

Autoencoders – denoising autoencoder

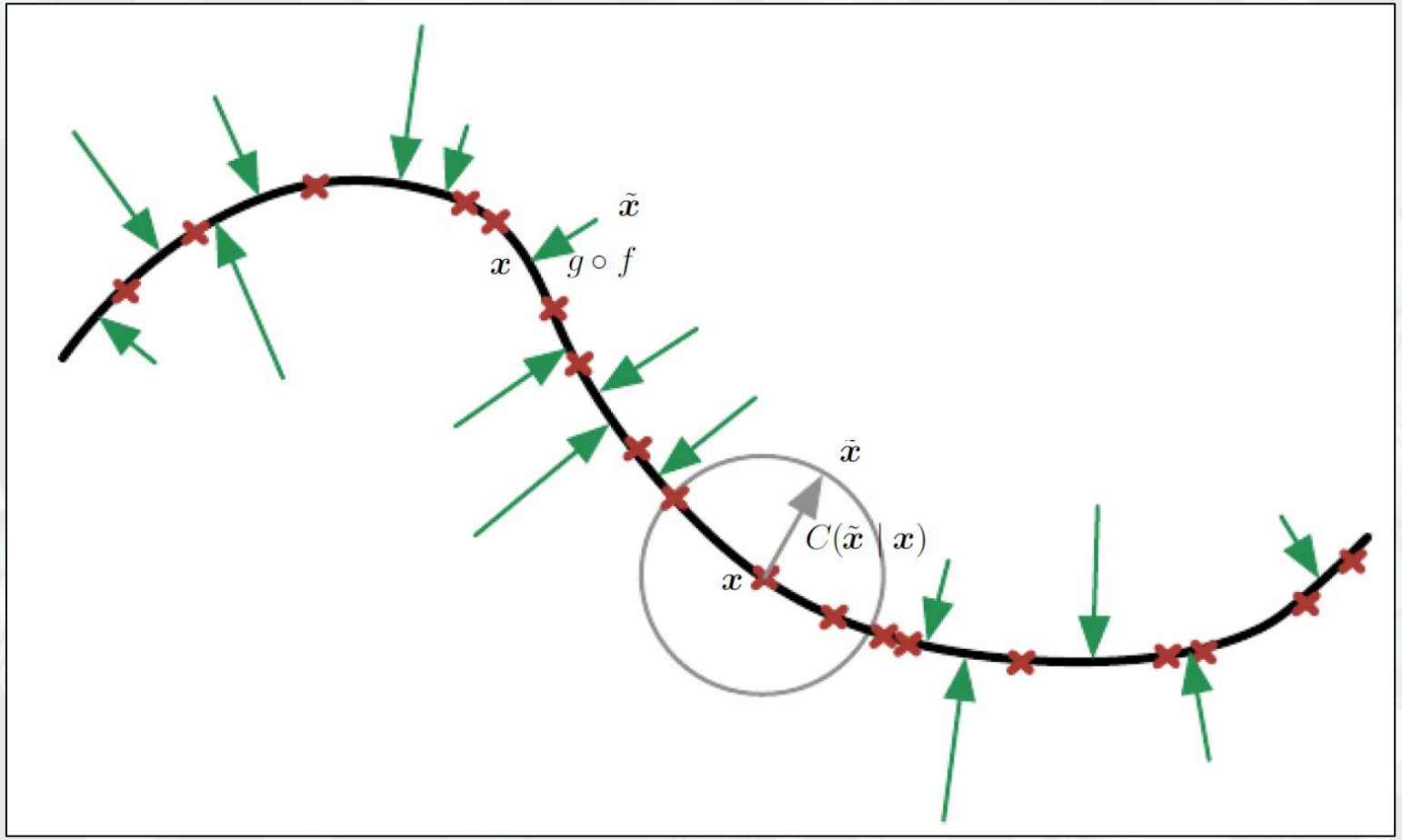
If an autoencoder can reconstruct the input, why couldn't it also reconstruct a slightly modified version of it?



Input: $\tilde{x} = x + n$
Output (reconstruction): x
Objective: learning to remove the noise n from the input
Optimization: minimizing the cost function

$$\mathcal{L} = L\left(x, g\left(f(\tilde{x})\right)\right) = L\left(x, g\left(f(x+n)\right)\right)$$

Autoencoders – denoising autoencoder



Training data space

Training samples

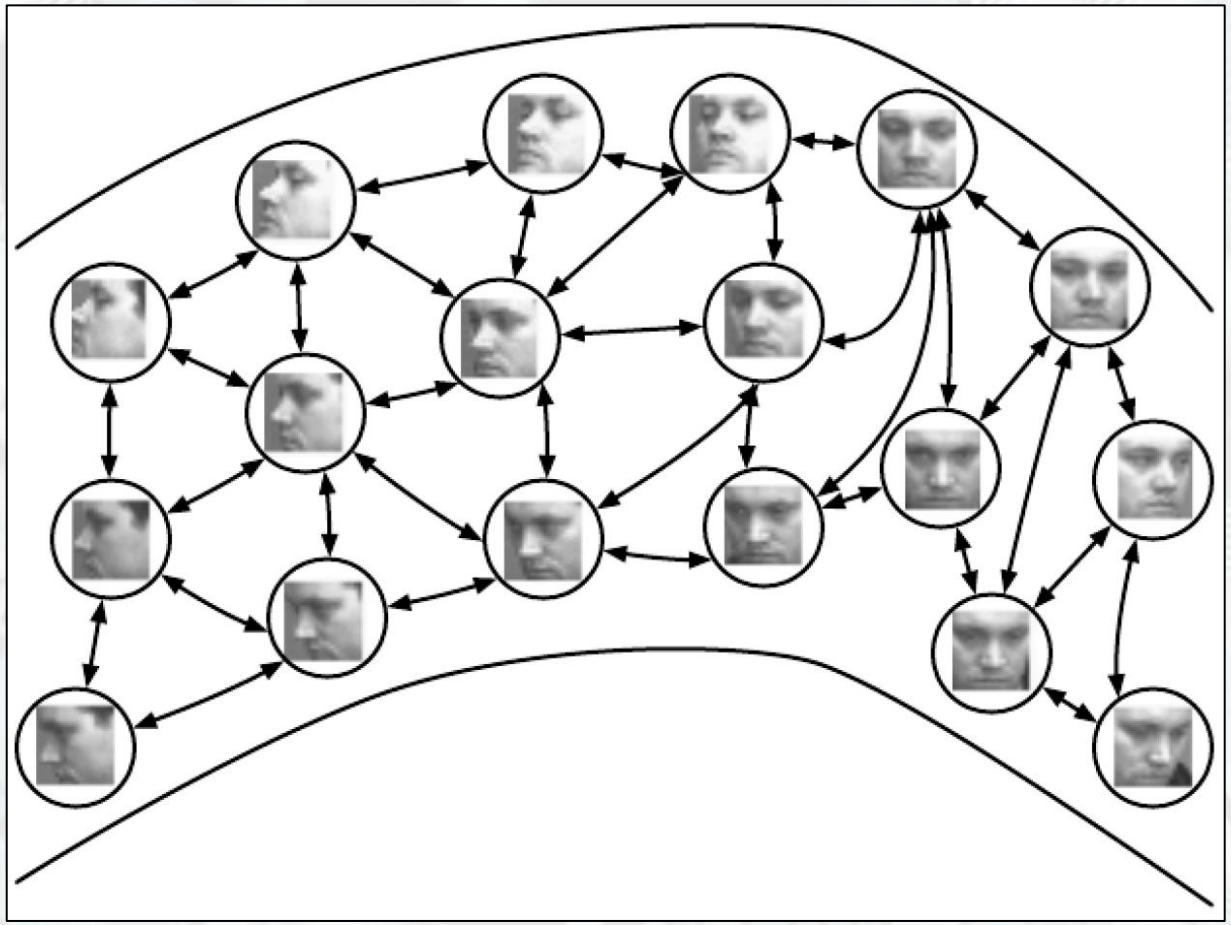
Data corruption

Data recovery

Autoencoders – learning the manifold

Learning the data space leads to the creation of a nearest neighbor graph:
- nodes = training samples;
- edges = nearest neighbor relationships.

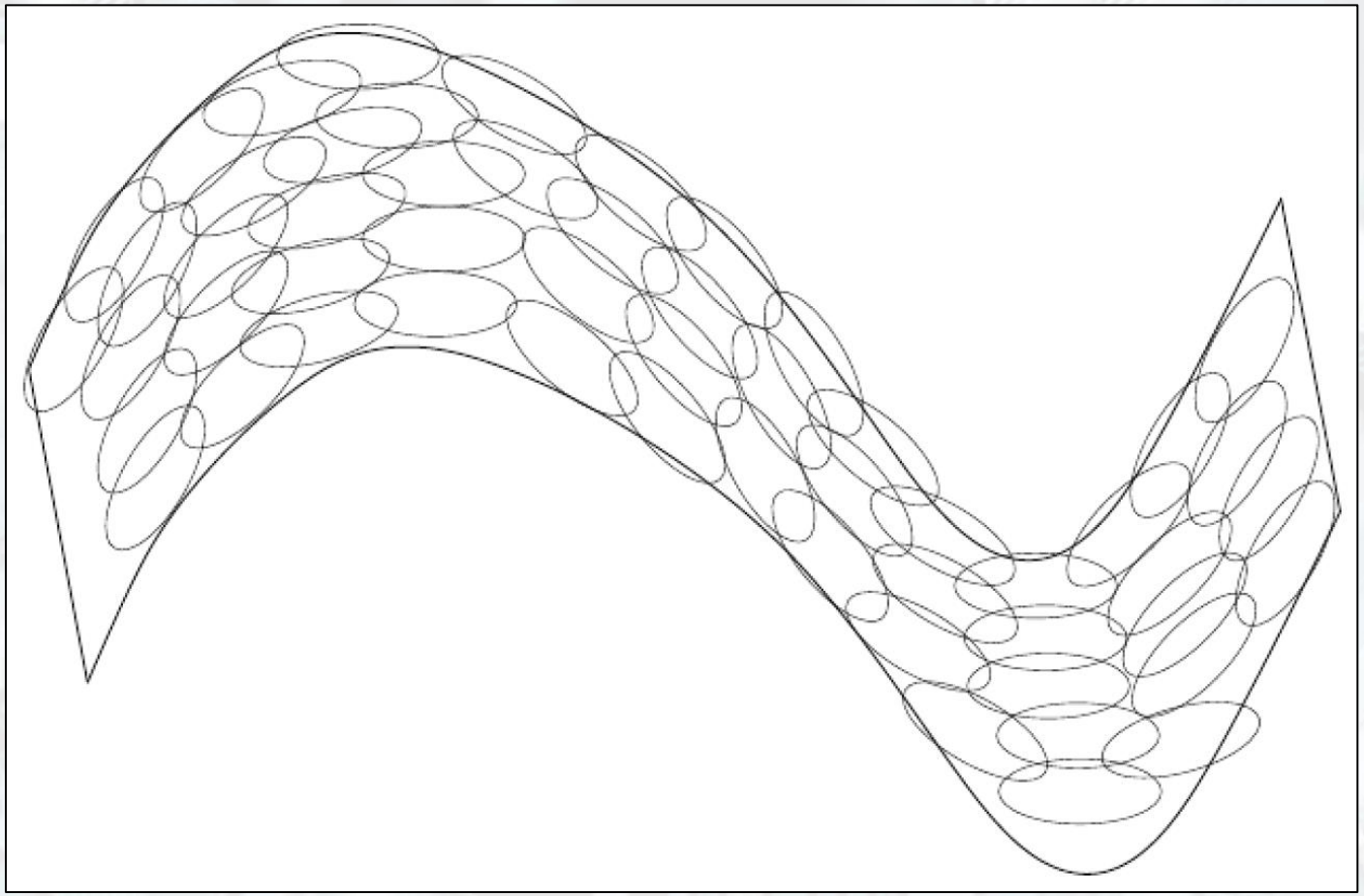
If there is enough training data, the entire data space can be learned and thus new samples can be generated by interpolation.



Sursă imagine: [1]

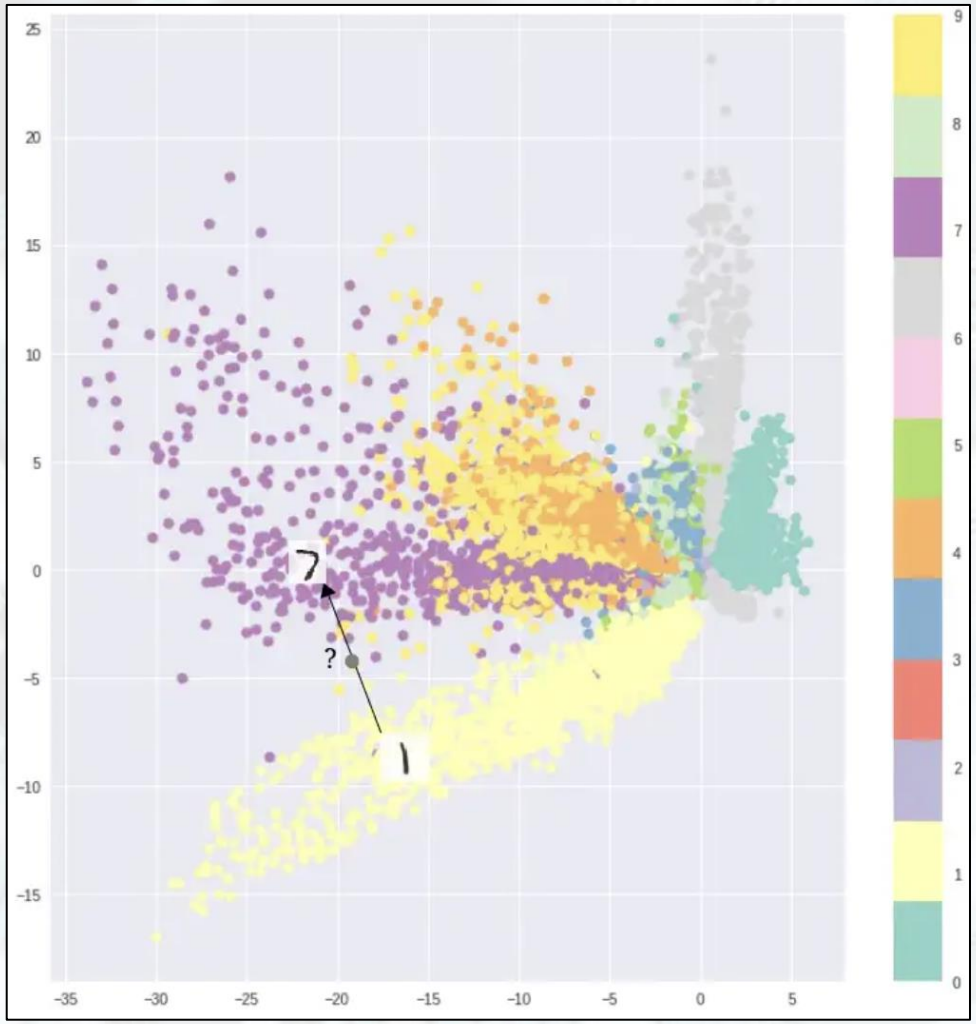
Autoencoders – learning the manifold

- Knowing the tangent planes at each location leads to combining them and estimating the probability density function of the data, which completely models the training data set.
- Each tangent plane is approximated by a Gaussian curve.
- In practice, the data space is very complex and can only be approximated (impossible to determine precisely).



Autoencoders – learning the manifold

- Designing descriptors in a 2D space;
- Training the autoencoder on MNIST Digits;
- Each figure belongs to a relatively well-defined cluster;
- We choose a point that is not on the graph - what image is decoded?
- Limitation: the decoder can process (replicate input) only descriptors it has seen during training – discrete value space. Decoding other values leads to meaningless images => not suitable for generating new examples.



Autoencoders – conclusions

- Autoencoders are very good at retrieving from a data space the information used in training;
- It involves training with a large number of examples to cover the entire data space (manifold);
- They are robust to a relatively small perturbation of the feature descriptor (of the embedding);
- They are not suitable for generating new examples.

Autoencoders – applications

1. Reducing data dimensionality
2. Information retrieval
3. Anomaly detection
4. Clustering
5. Denoising
6. Inpainting

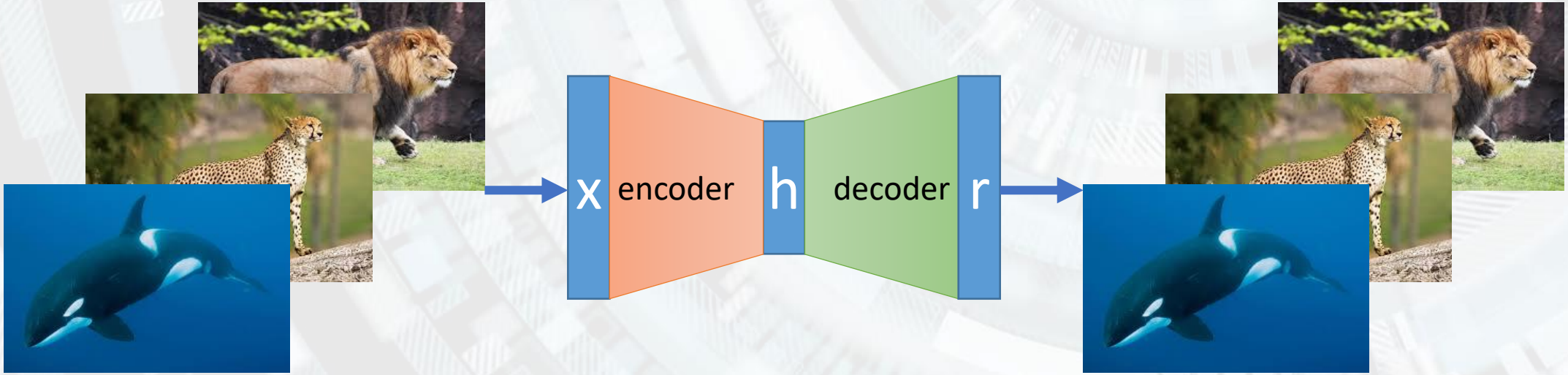
Autoencoders – applications

1. Reducing data dimensionality

Autoencoders – applications

2. Information retrieval

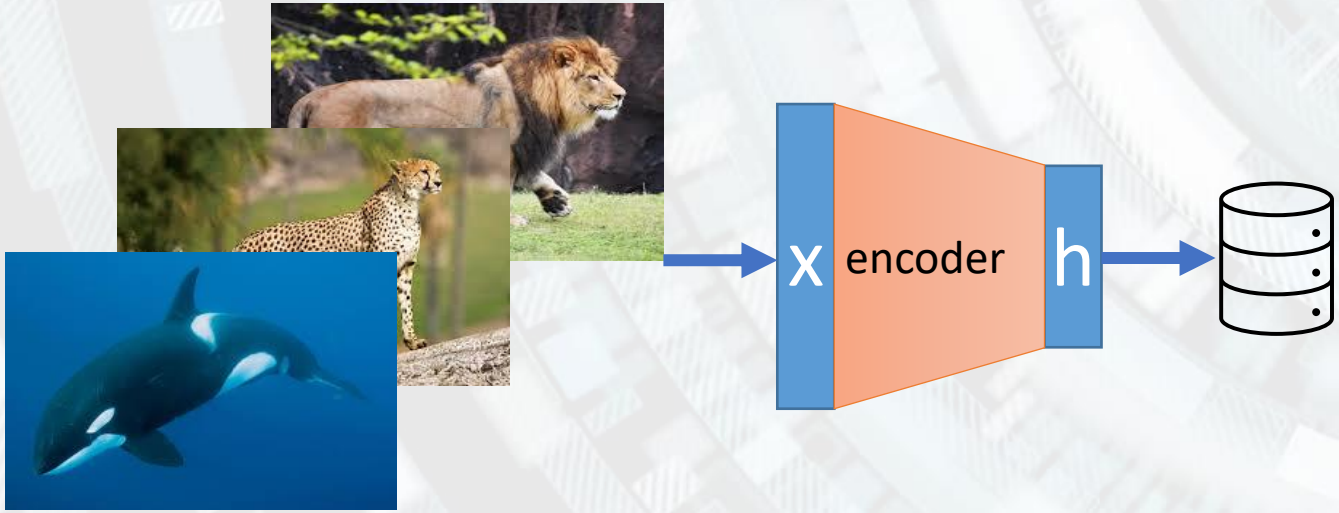
- Training the autoencoder on the train dataset



Autoencoders – applications

2. Information retrieval

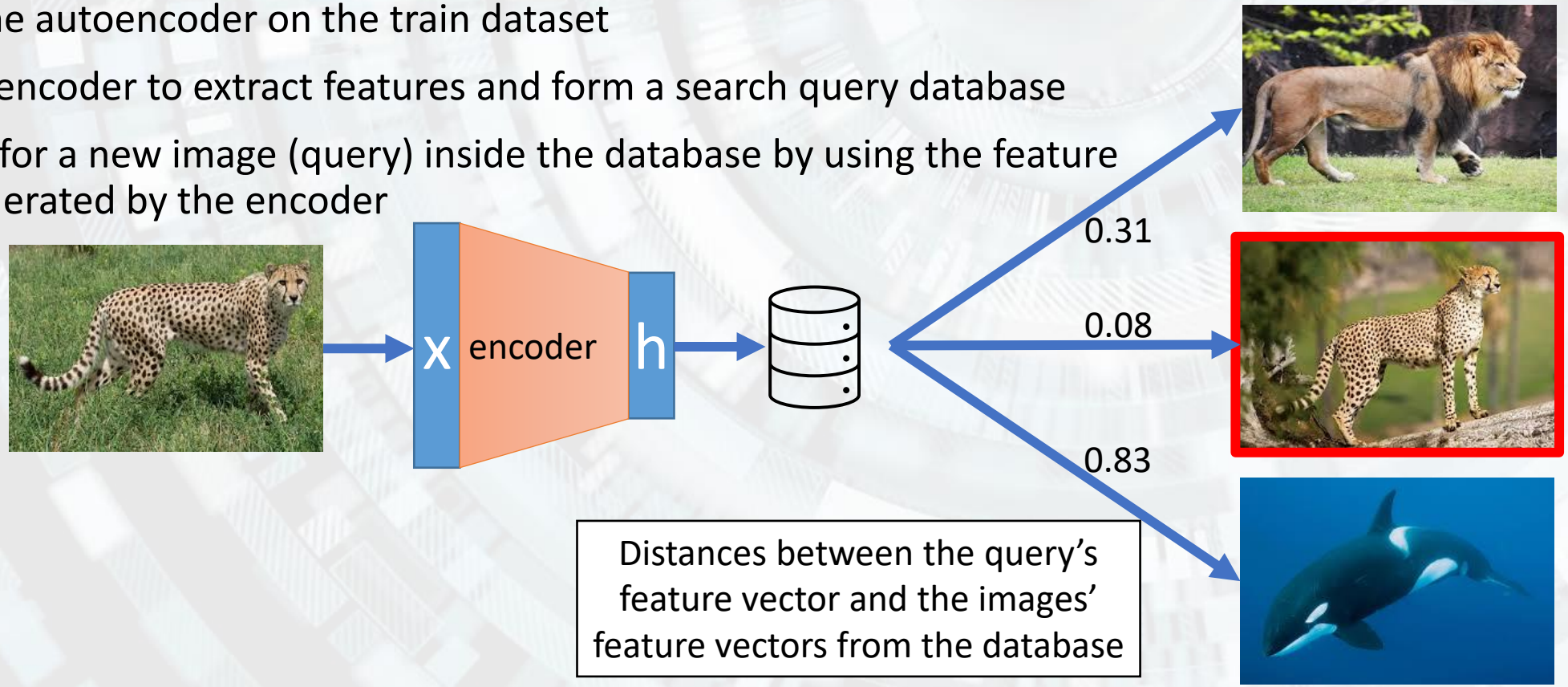
- Training the autoencoder on the train dataset
- Using the encoder to extract features and form a search query database



Autoencoders – applications

2. Information retrieval

- Training the autoencoder on the train dataset
- Using the encoder to extract features and form a search query database
- Searching for a new image (query) inside the database by using the feature vector generated by the encoder

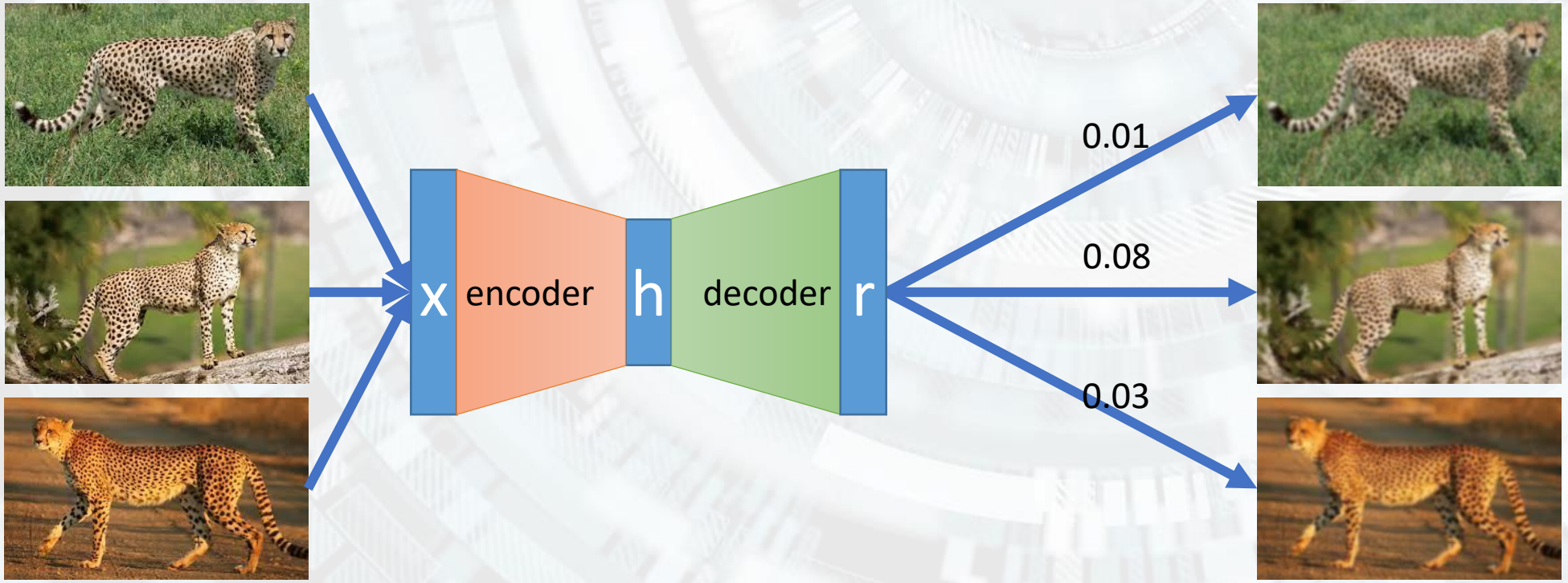


Autoencoders – applications

3. Anomaly detection

- Training the autoencoder with the train set, without anomalies

Outputs with low reconstruction errors

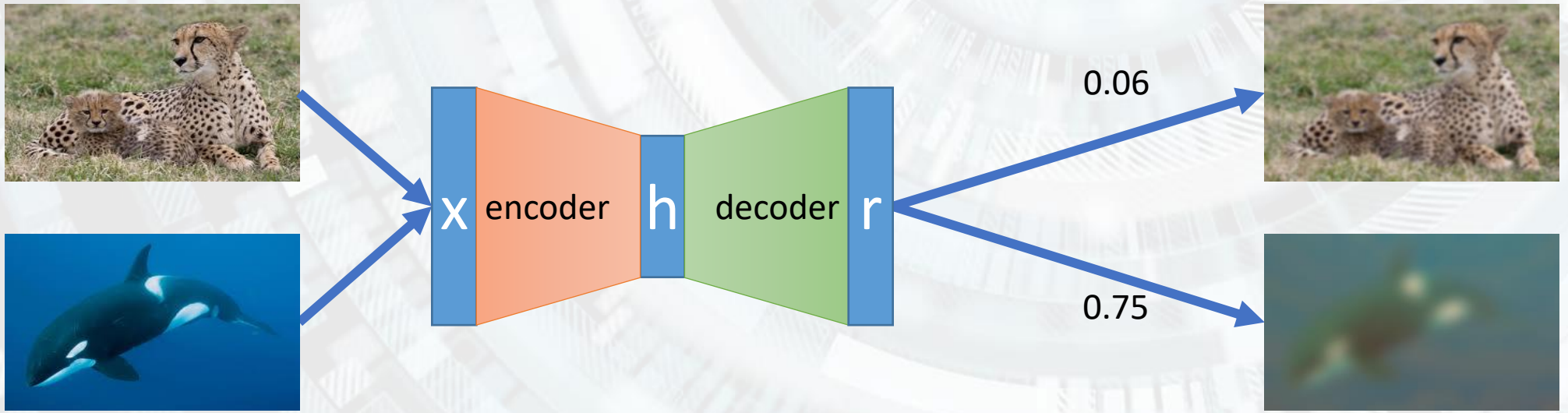


Autoencoders – applications

3. Anomaly detection

- Training the autoencoder with the train set, without anomalies
- Testing the autoencoder with diverse images (with/without anomalies)

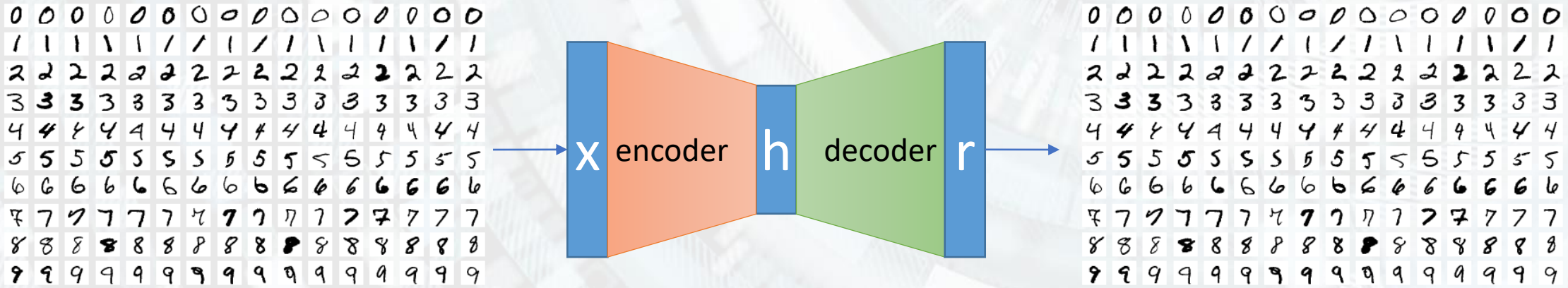
High reconstruction errors indicates an anomaly



Autoencoders – applications

4. Clustering

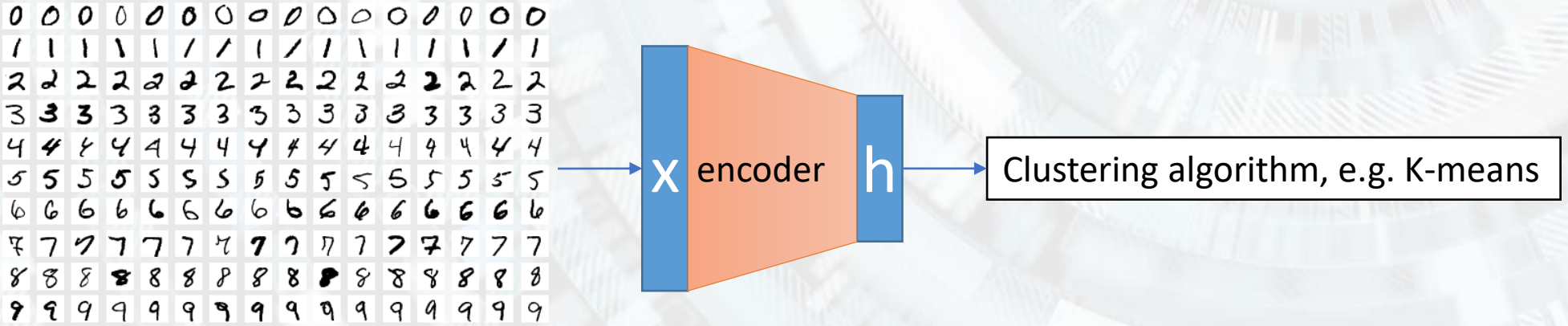
- Training the autoencoder on the train set



Autoencoders – applications

4. Clustering

- Training the autoencoder on the train set
- Using the encoder to generate feature vectors that will be used by clustering algorithms, e.g. k-means.



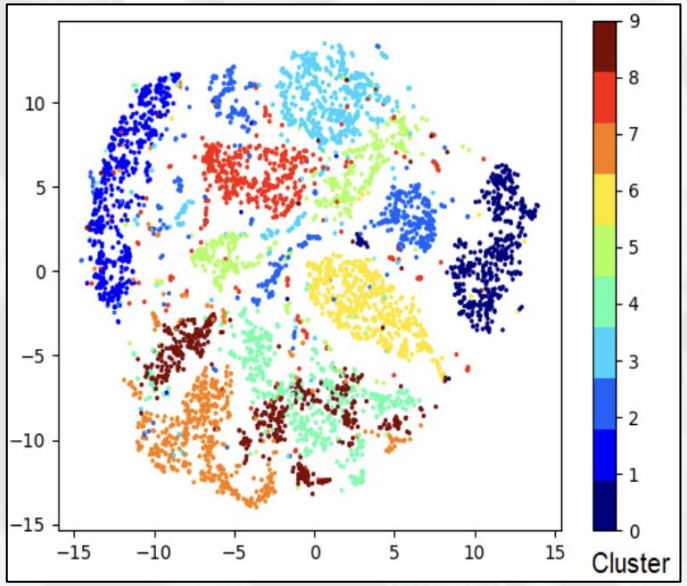
Sursă imagine: [3]

Autoencoders – applications

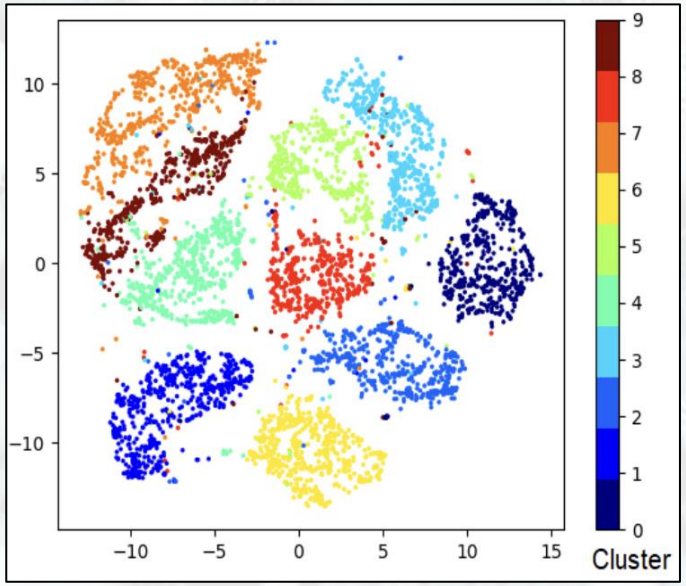
4. Clustering

- Training the autoencoder on the train set
- Using the encoder to generate feature vectors that will be used by clustering algorithms, e.g. k-means.

K-means run in pixel space



vs

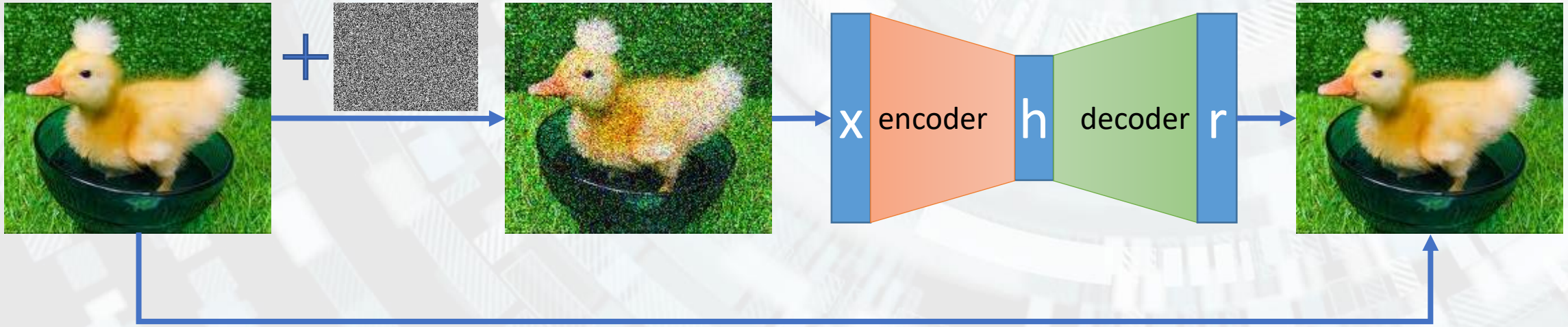


K-means run in latent space

Autoencoders – applications

5. Denoising

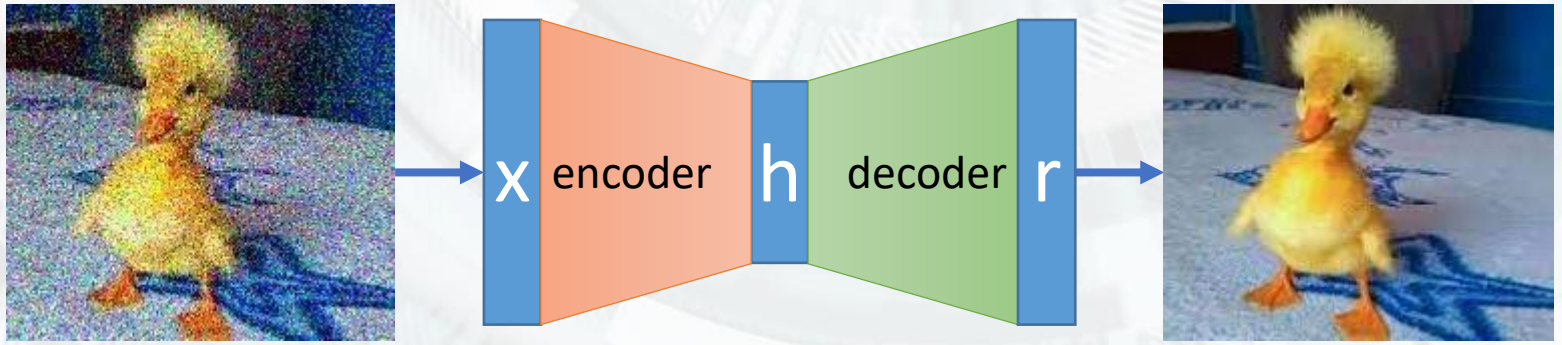
- Training the autoencoder with the train set under the format input = original + noise, output = original.



Autoencoders – applications

5. Denoising

- Training the autoencoder with the train set under the format input = original + noise, output = original.
- Testing the autoencoder by feeding it noisy images. The autoencoder learns to eliminate the type of noise it has been trained with.

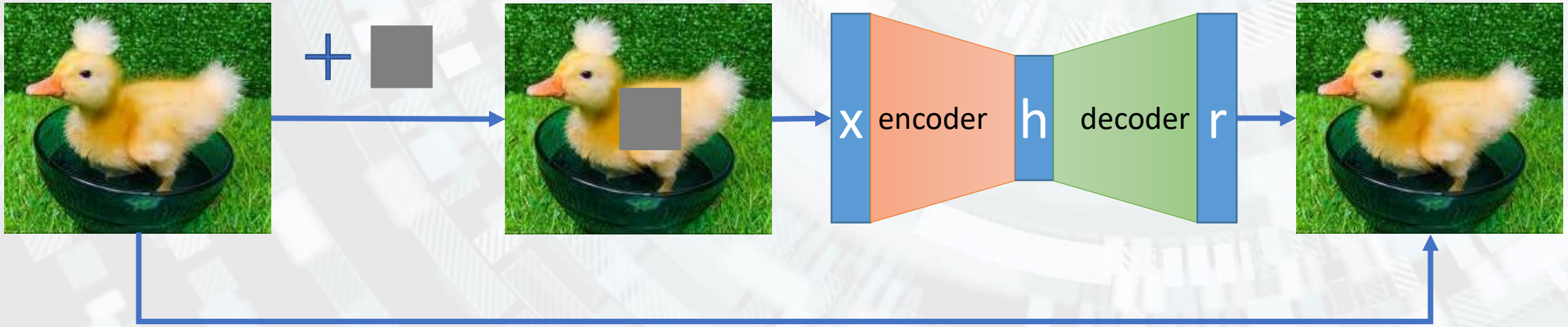


Let's test it out – unit #8

Autoencoders – applications

6. Inpainting

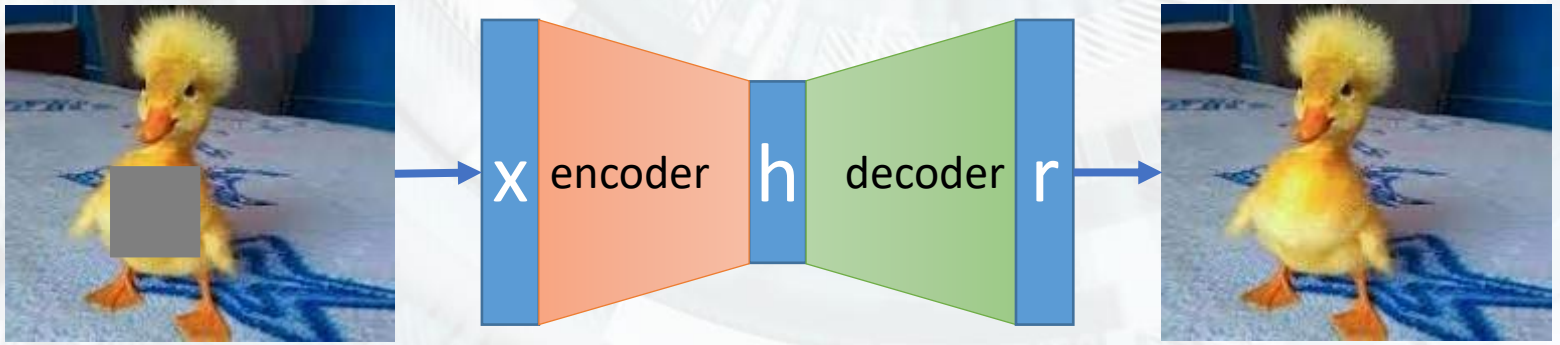
- Training the autoencoder with the train set under the format input = original + occlusion, output = original. It is a particular type of denoising.



Autoencoders – applications

6. Inpainting

- Training the autoencoder with the train set under the format input = original + occlusion, output = original. It is a particular type of denoising.
- Testing the autoencoder with images with occlusions. The autoencoder learns to retrieve the information behind the occlusion.

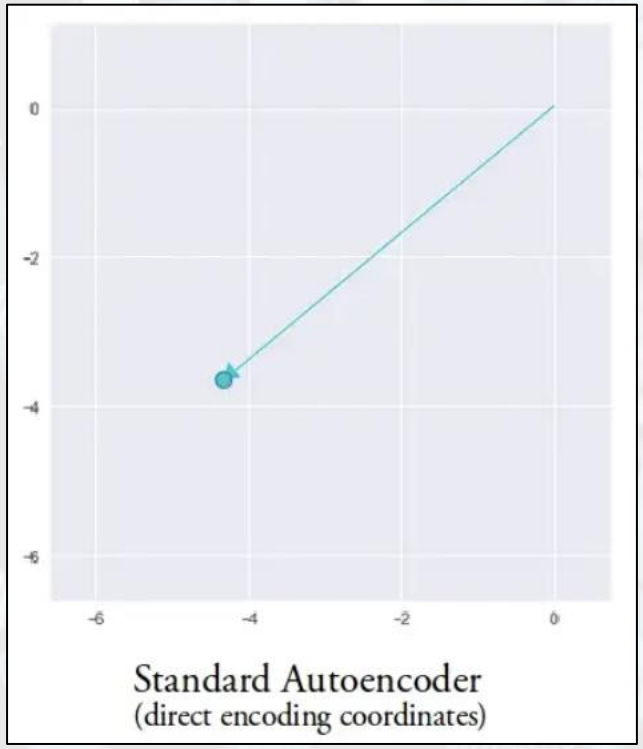


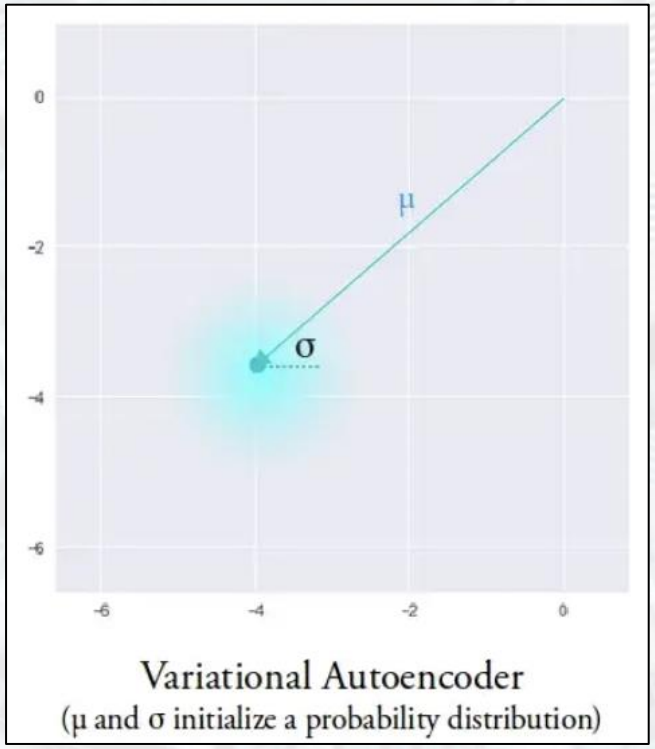
M4.3. Variational Autoencoders (VAEs)

Image: Irhum Shafkat @ towardsdatascience.com

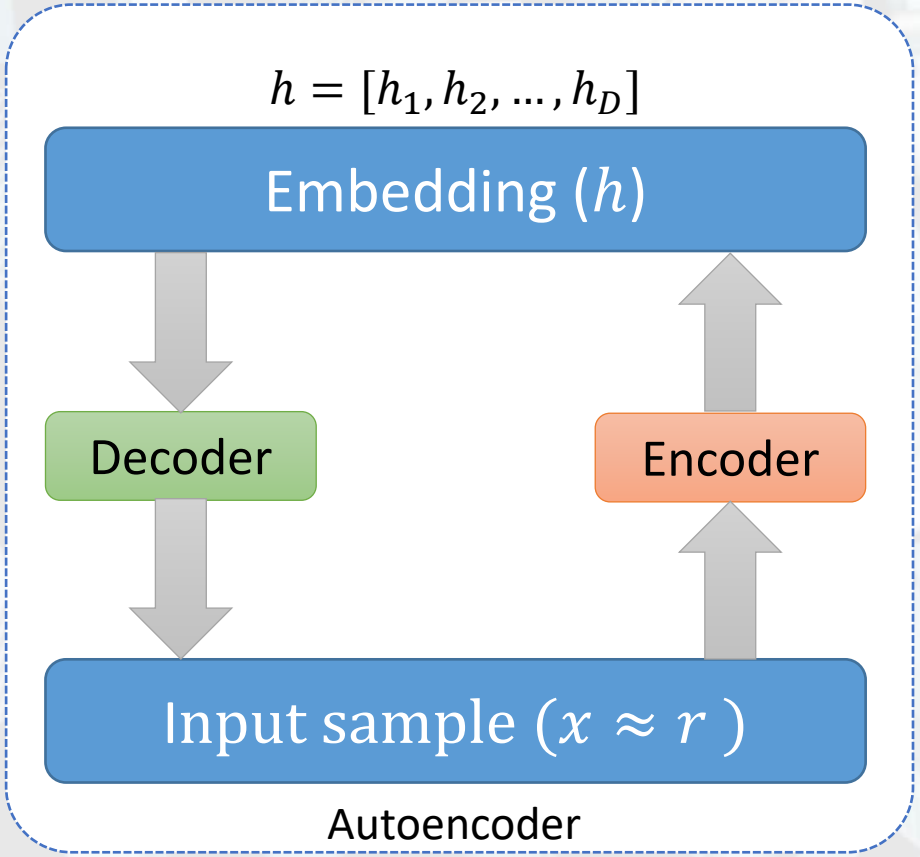
Variational Autoencoders

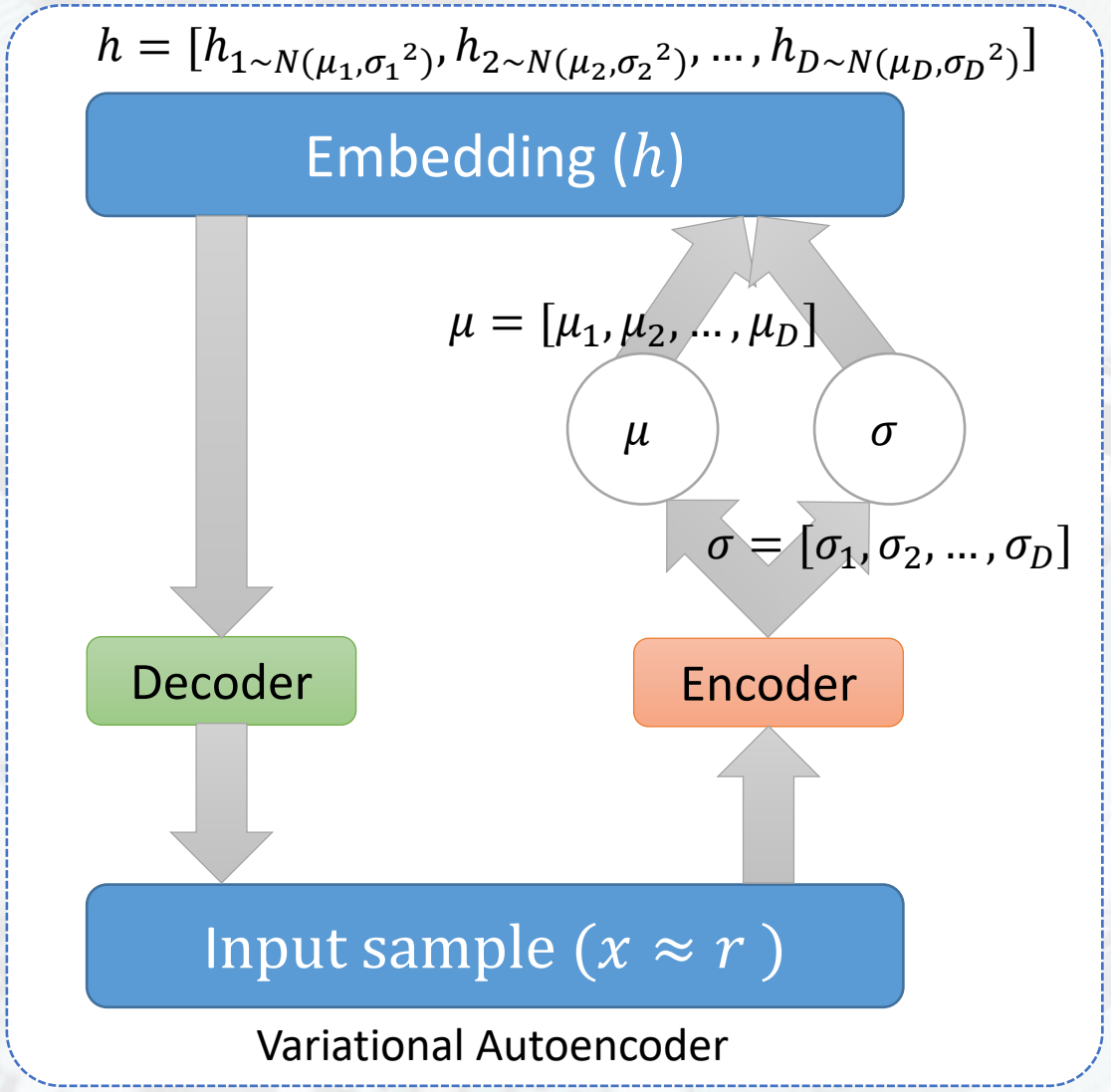
VAEs take the idea of the autoencoder and they adapt it to generation. How do they do that?





Variational Autoencoders





Variational Autoencoders

$$h = [h_{1\sim N(\mu_1,{\sigma_1}^2)}, h_{2\sim N(\mu_2,{\sigma_2}^2)}, \dots, h_{D\sim N(\mu_D,{\sigma_D}^2)}]$$

- The descriptor is generated as a sample from a normal distribution with mean μ and standard deviation σ => the same training example will generate different descriptors, but from the same probability distribution;
- μ controls the area where the distribution will be centered;
- σ controls the area on which the distribution will unfold;
- The decoder will associate a neighborhood (assigned to the normal distribution) with a single output image.
- Continuous association, not discrete (as with autoencoder)

Variational Autoencoders

$$h = [h_{1\sim N(\mu_1,{\sigma_1}^2)}, h_{2\sim N(\mu_2,{\sigma_2}^2)}, \dots, h_{D\sim N(\mu_D,{\sigma_D}^2)}]$$

- Problem: sampling from a distribution probability is not differentiable (you cannot run backpropagation on such an operation) => the model cannot learn the descriptor h.
- Solution: reparameterization.

$$h\sim N(\mu, \sigma^2) \leftrightarrow$$
$$h = \mu + \sigma * \varepsilon, \qquad \varepsilon\sim N(0,1)$$

- The sampling part is moved from the descriptor to ε, outside the VAE network.

Variational Autoencoders – reparametrizare

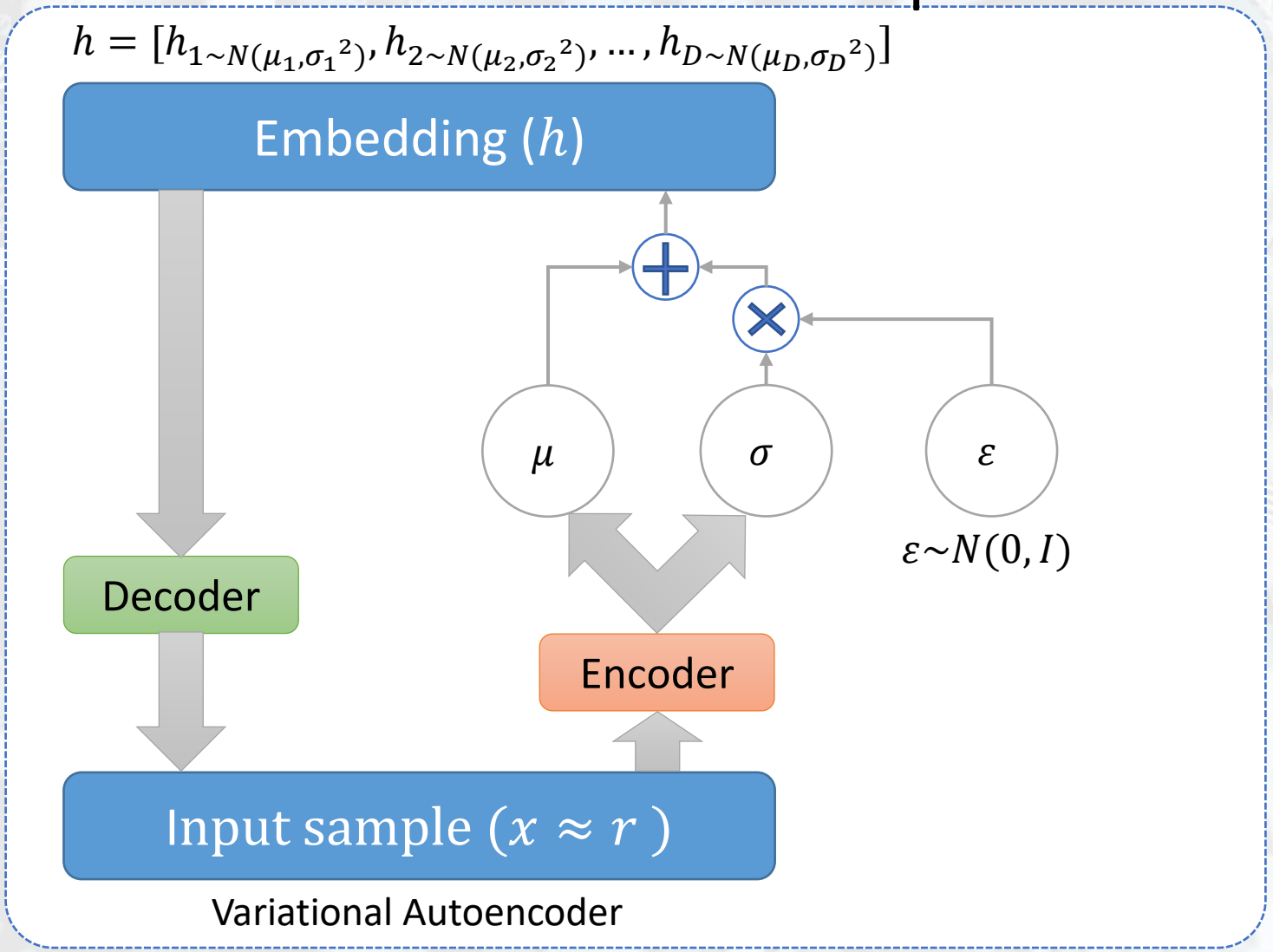
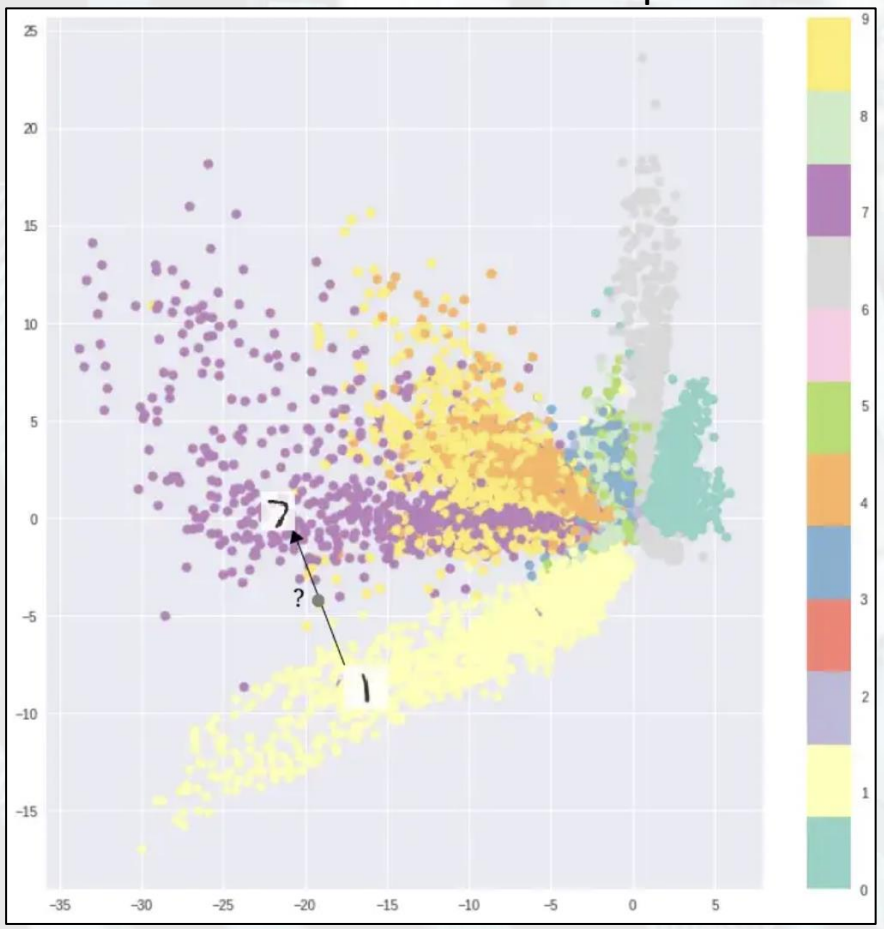


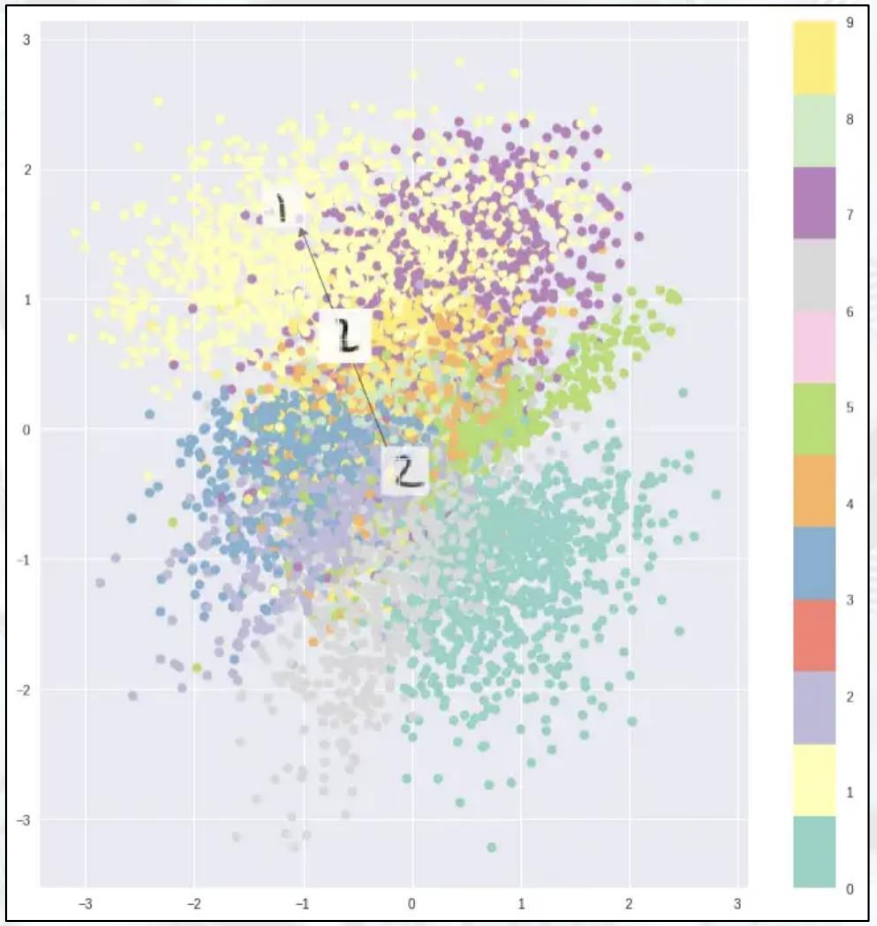
Image: Irhum Shafkat @ towardsdatascience.com

Variational Autoencoders

Autoencoder – discrete space



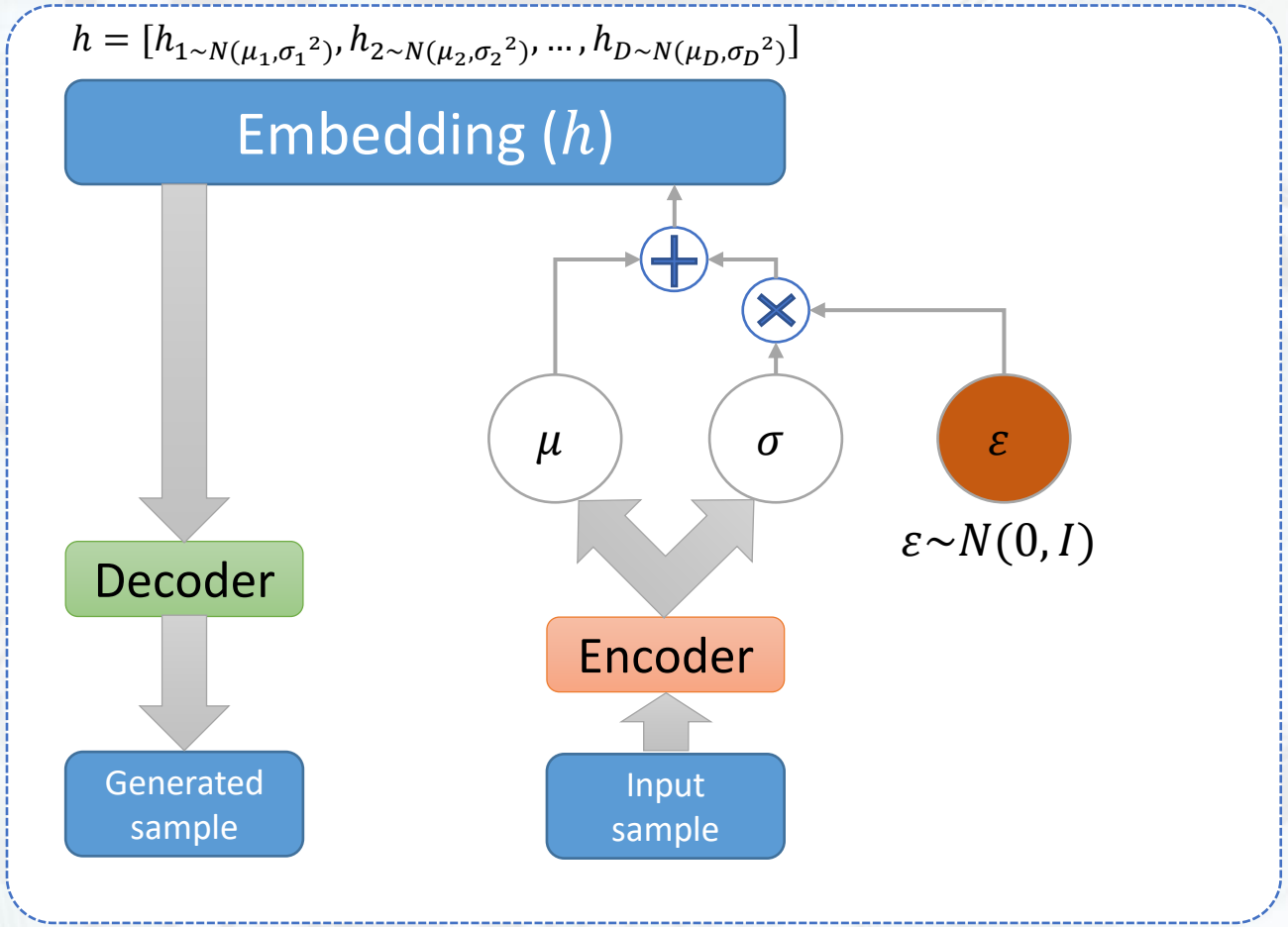
Variational autoencoder – continuous space



Variational Autoencoders – applications

1. Generate new samples:

- Train the VAE on the train subset.
- Process an input sample and obtain μ și σ.
- Sample a random value ε.
- From μ, σ and ε generate the embedding h.
- h is decoded into the sample space by the decoder. The resulting sample is a new sample (with probability almost equal to 1), not encountered in the training database.



Variational Autoencoders – applications

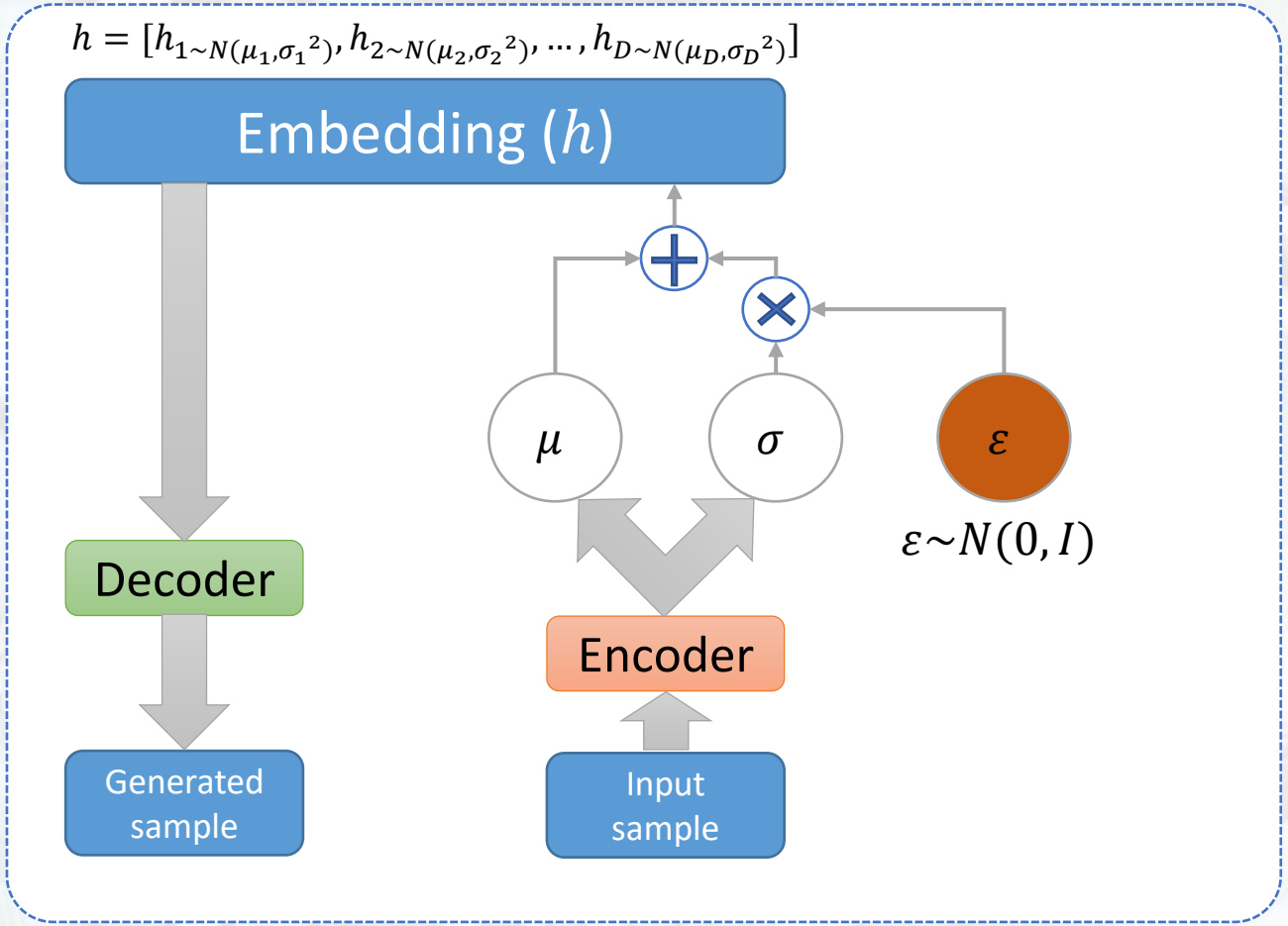
2. Interpolate samples:

- Train the VAE on the train subset.
- Process 2 test samples and obtain their respective features h_{in_1} și h_{in_2}.
- Set a number of interpolation steps N, e.g. N = 10.
- Generate latent interpolation vectors

$$h_i = h_{in\_1} + \left(h_{in_2} - h_{in_1}\right) * \frac{i}{N}, \mathrm{i} = 0 \dots N,$$

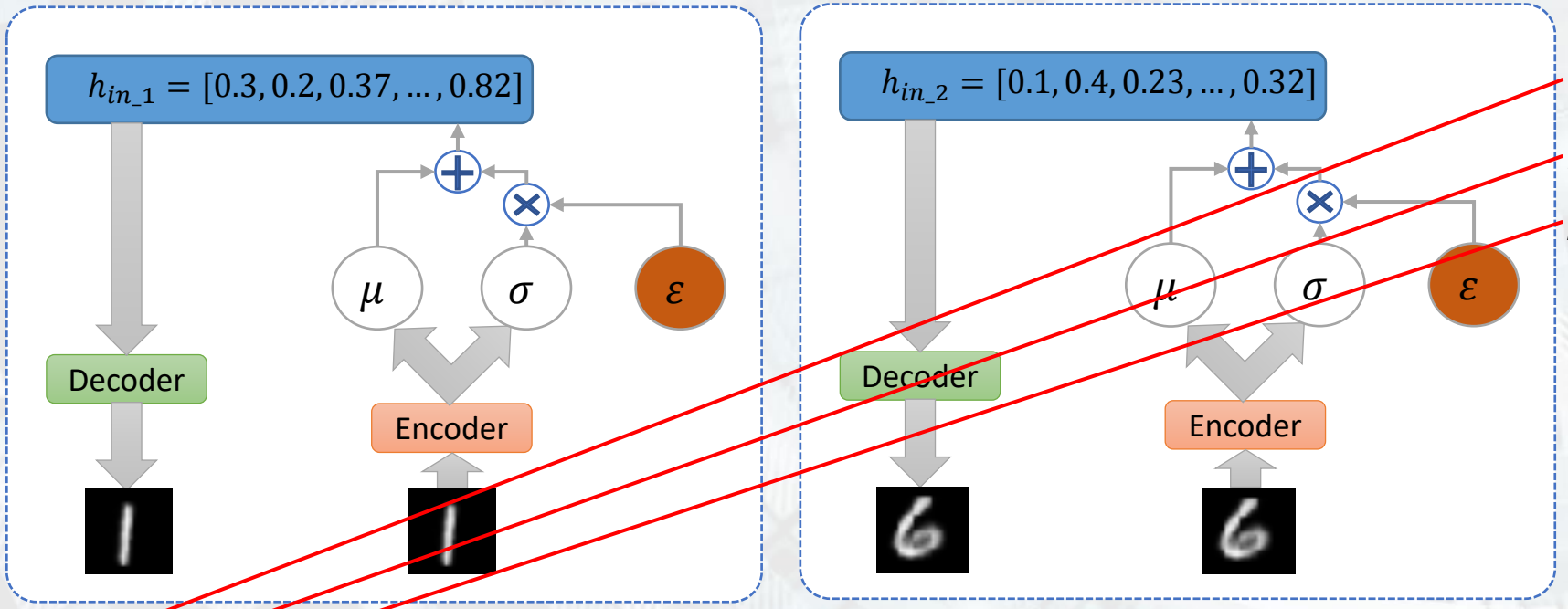
where N is the number of interpolation steps.

- Each latent vector is decoded under a new sample.



Variational Autoencoders – applications

2. Interpolate samples:



$h_1 = [0.3, 0.2, 0.37, \dots, 0.82]$

$h_2 = [0.29, 0.21, 0.363, \dots, 0.795]$

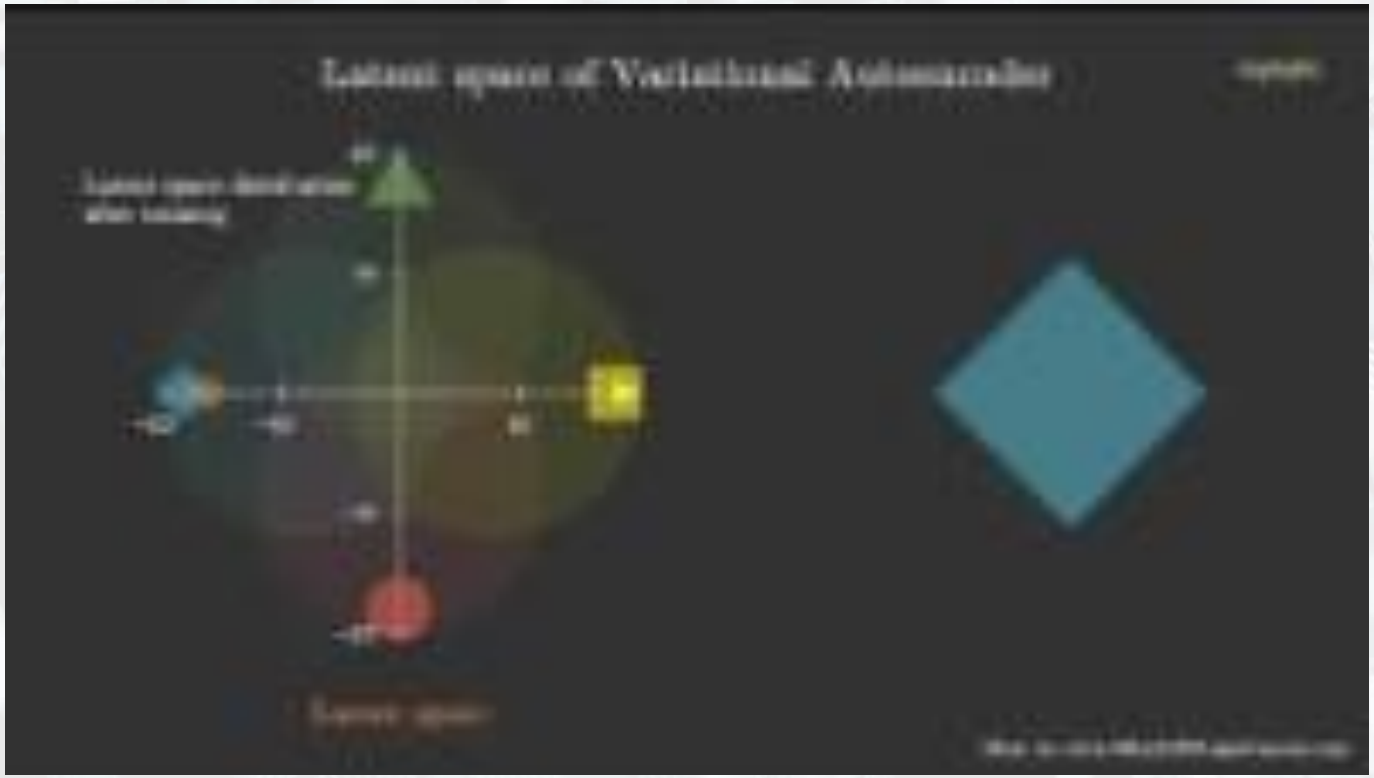
$h_3 = [0.28, 0.22, 0.356, \dots, 0.77]$

$\vdots$

$h_{20} = [0.1, 0.4, 0.23, \dots, 0.32]$

Variational Autoencoders – applications

2. Interpolate samples :

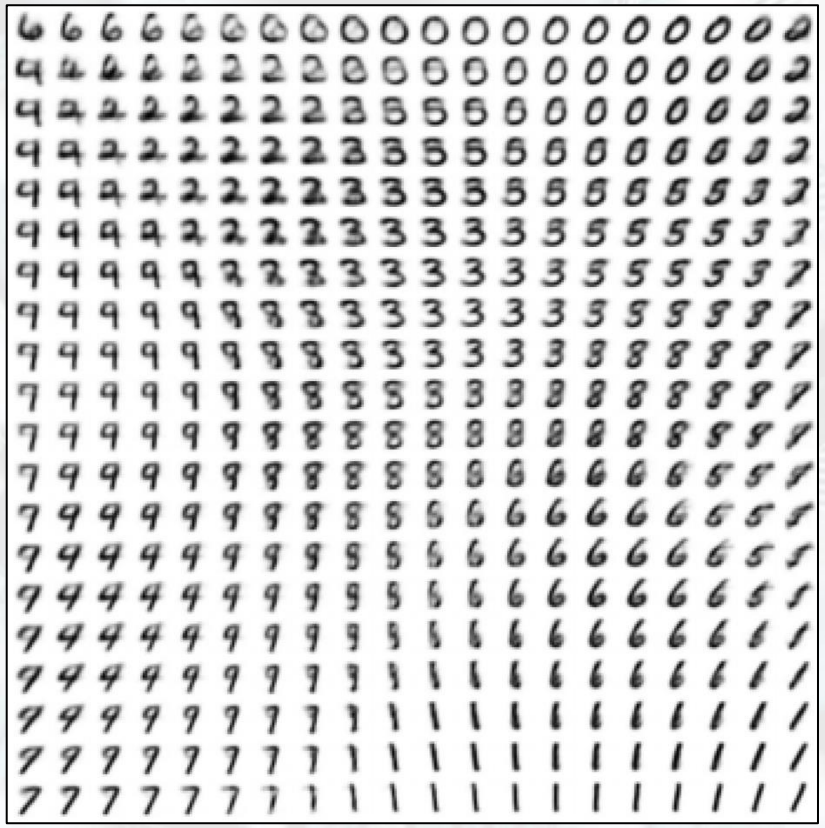


https://www.youtube.com/watch?v=sV2FOdGqlX0&t=1s&ab_channel=AqeelAnwar

Variational Autoencoders – applications

Let's test it out – unit #9

2. Interpolate samples :



Variational Autoencoders – applications

3. Modify specific features:

Common features?

Smile

Glasses

Variational Autoencoders – applications

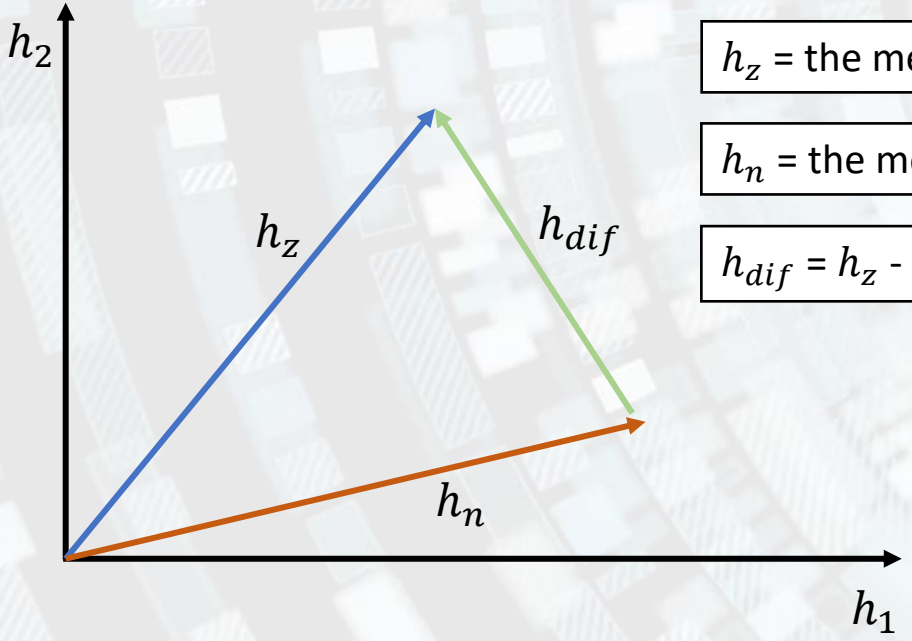
3. Modify specific features:

- Valid only for some datasets, where the specific features are annotated.

	5_o_Clock_Shadow	Arched_Eyebrows	Attractive	Bags_Under_Eyes	Bald	Bangs	Big_Lips	Big_Nose	Black_Hair	Blond_Hair	Blurry	Brown_Hair	Bushy_Eyebrows	Chubby
000001.jpg	-1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1
000002.jpg	-1	-1	-1	1	-1	-1	-1	1	-1	-1	-1	1	-1	-1
000003.jpg	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	1	-1	-1	-1
000004.jpg	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
000005.jpg	-1	1	1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1
000006.jpg	-1	1	1	-1	-1	-1	1	-1	-1	-1	-1	1	-1	-1
000007.jpg	1	-1	1	1	-1	-1	1	1	1	-1	-1	-1	1	-1
000008.jpg	1	1	-1	1	-1	-1	1	-1	1	-1	-1	-1	-1	-1
000009.jpg	-1	1	1	-1	-1	1	1	-1	-1	-1	-1	-1	-1	-1
000010.jpg	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
000011.jpg	-1	-1	1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1
000012.jpg	-1	-1	1	1	-1	-1	-1	-1	1	-1	-1	-1	1	-1
000013.jpg	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1
000014.jpg	-1	1	-1	-1	-1	-1	-1	1	1	-1	-1	-1	1	-1
000015.jpg	1	-1	-1	1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1
000016.jpg	1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
000017.jpg	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1
000018.jpg	-1	1	-1	-1	-1	-1	-1	1	-1	1	-1	-1	-1	-1
000019.jpg	-1	1	1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1
000020.jpg	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	1
000021.jpg	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1
000022.jpg	-1	1	-1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	-1
000023.jpg	1	-1	1	-1	-1	-1	-1	1	-1	-1	-1	1	-1	-1
000024.jpg	-1	1	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	-1
000025.jpg	1	-1	-1	1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1

Variational Autoencoders – applications

3. Modify specific features:

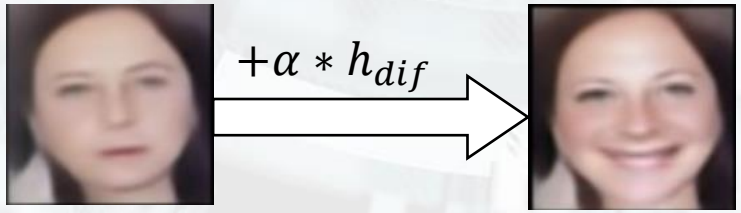


h_z = the mean of all feature vectors of samples where we have people smiling

h_n = the mean of all feature vectors of samples where we have people not smiling

h_{dif} = h_z - h_n = the feature vector encoding the smile

$$h_{nou} = h_{original} + \alpha * h_{dif}$$



Variational Autoencoders

Limitation: although VAEs are capable of generating completely new images, they are usually blurry. This is due to the noise introduced by the cost function, which averages out the differences between pixels.

Solution: using a system that also takes into account the "realism" of the image.

M4.4. Generative Adversarial Networks (GANs)

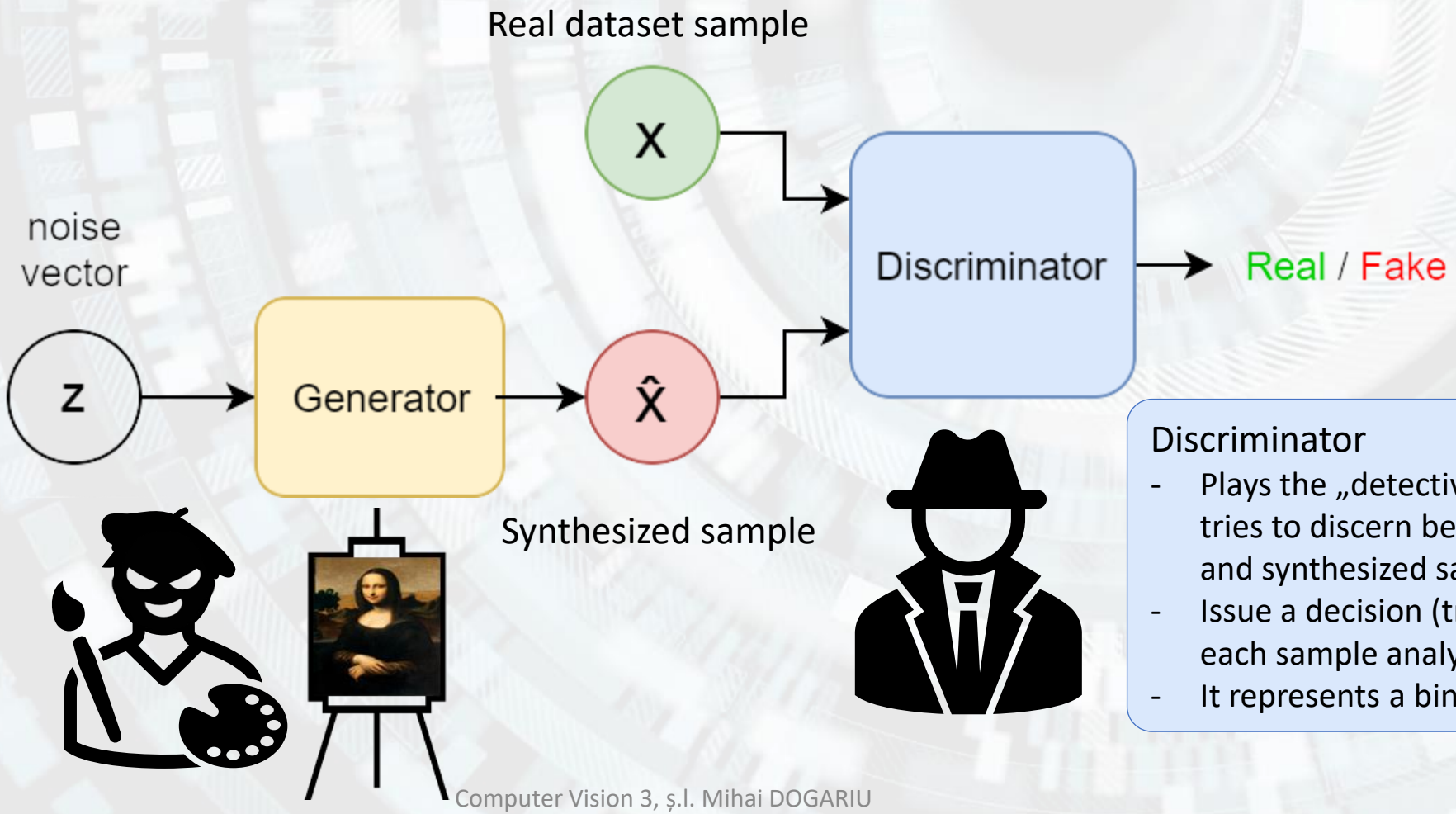
Generative Adversarial Networks

Generative adversarial networks are a class of neural networks specialized in generating realistic examples similar to the training data.

- The model discovers the characteristics of the probability distribution of the training data and uses them to generate new samples.
- The ensemble is composed of 2 neural networks: a generator and a discriminator.
  - The generator is tasked with generating new examples.
  - The discriminator has the task of detecting synthetically generated examples.
- GAN is based on game theory: zero-sum game.

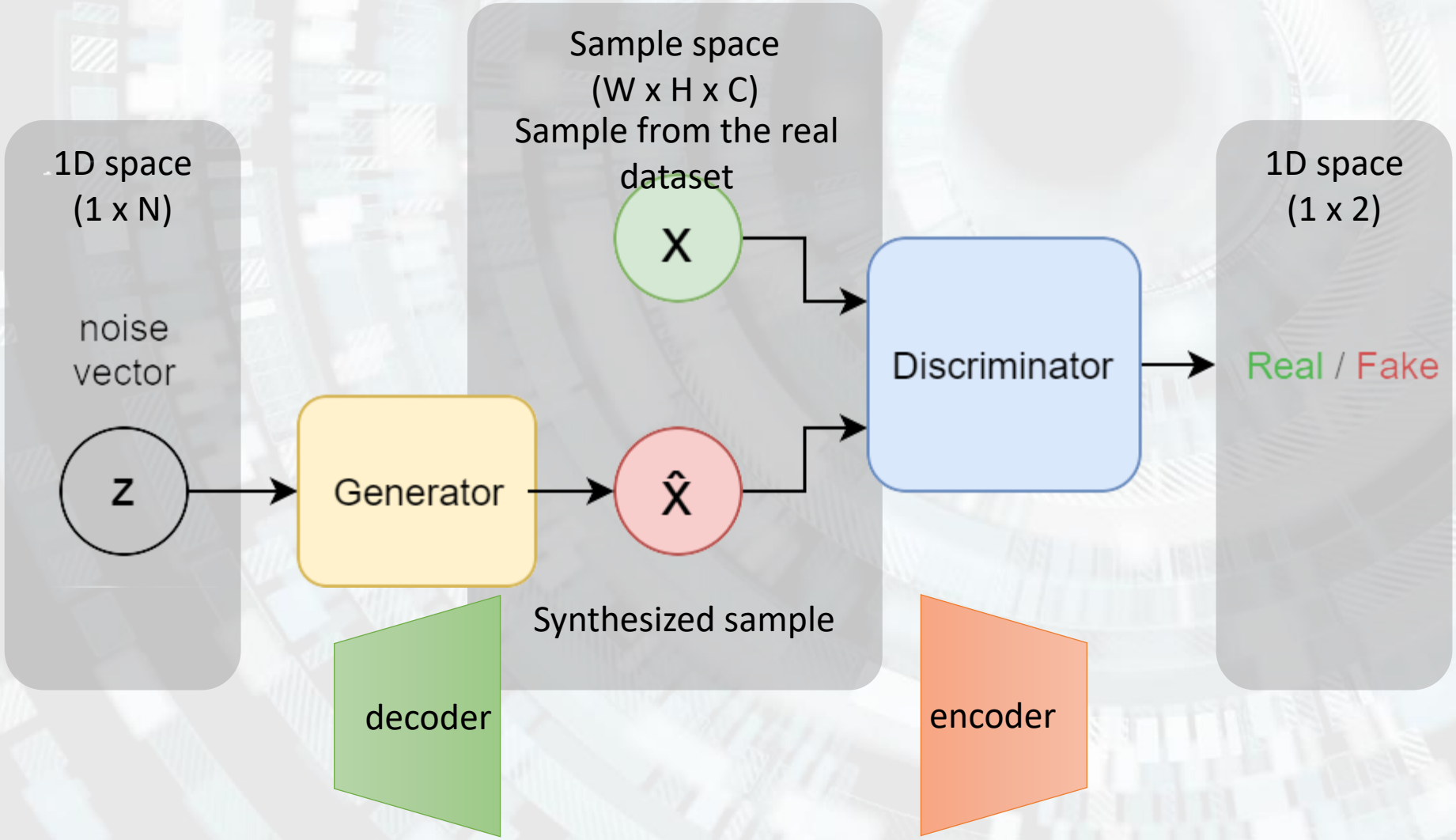
Generative Adversarial Networks

Generator
- Plays the „forger” role – tries to fool the discriminator;
- It takes the noise vector (normal, Gaussian) and transforms it into the data space;
- Generate data as plausible as possible (as similar to real data as possible).



Discriminator
- Plays the „detective” role – tries to discern between real and synthesized samples;
- Issue a decision (true/false) for each sample analyzed;
- It represents a binary classifier.

Generative Adversarial Networks



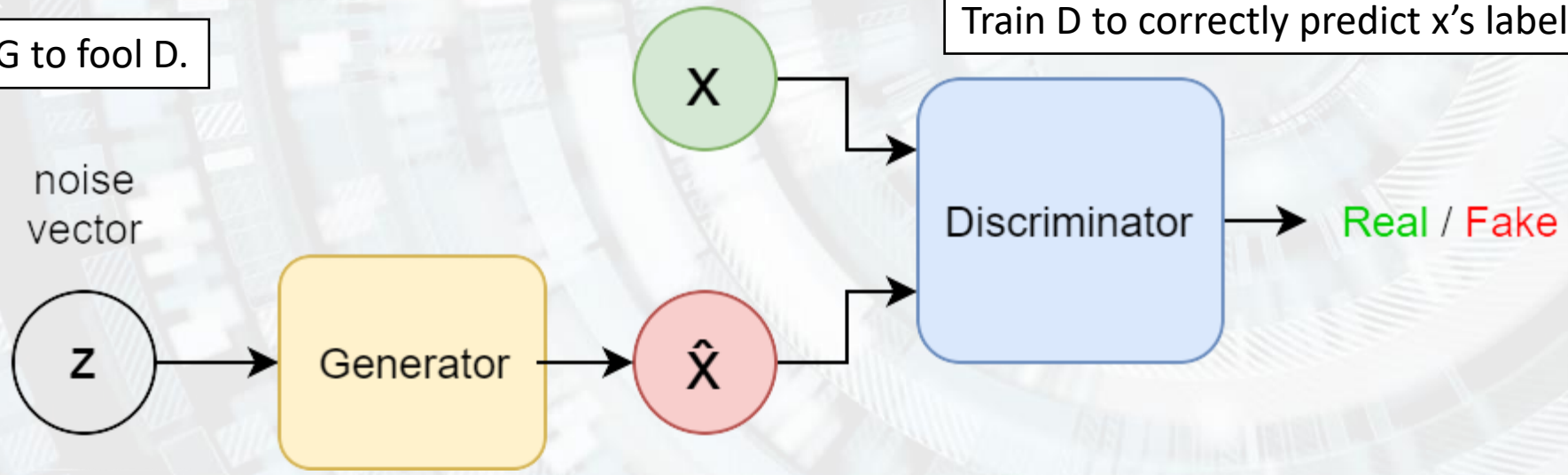
Generative Adversarial Networks

$X \sim p_{data}$

Sample from the real dataset

Train G to fool D.

Train D to correctly predict x's label.



Synthesized sample

$p_z(z)$ $G(z;\theta_g)$ $G(z) = \hat{X} \sim p_g$ $D(x;\theta_d)$ $D(x)$

Training is done by following a min-max game:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

Generative Adversarial Networks

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

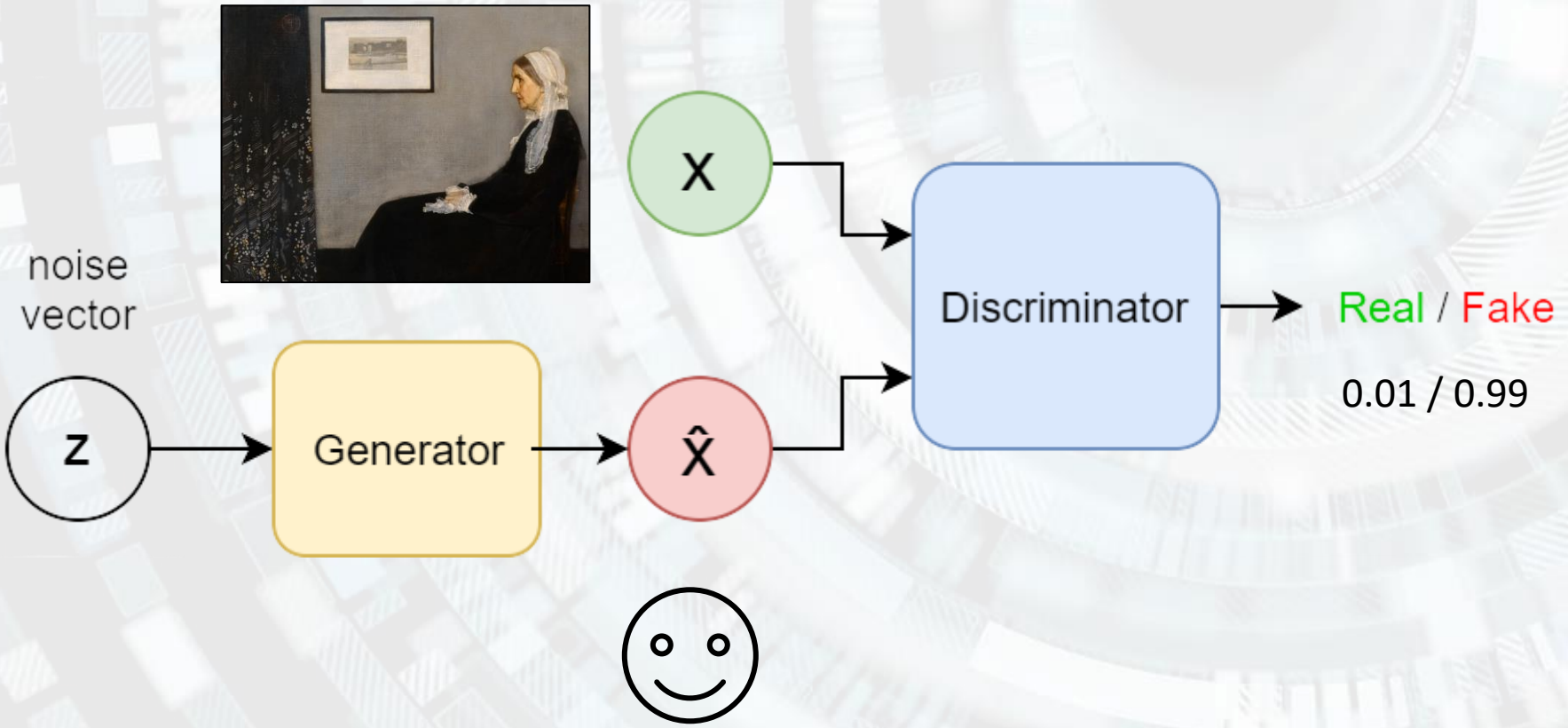
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

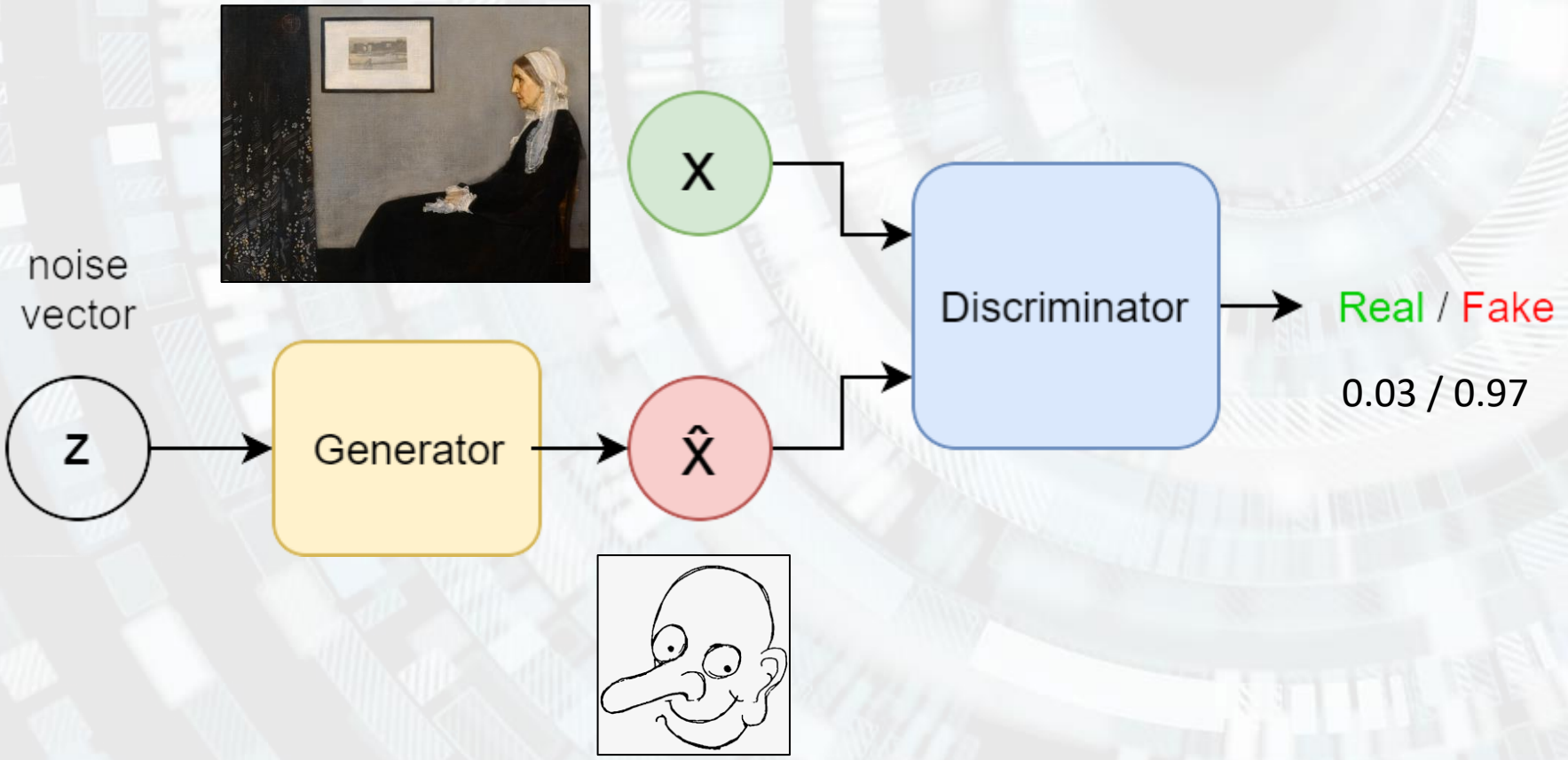
end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

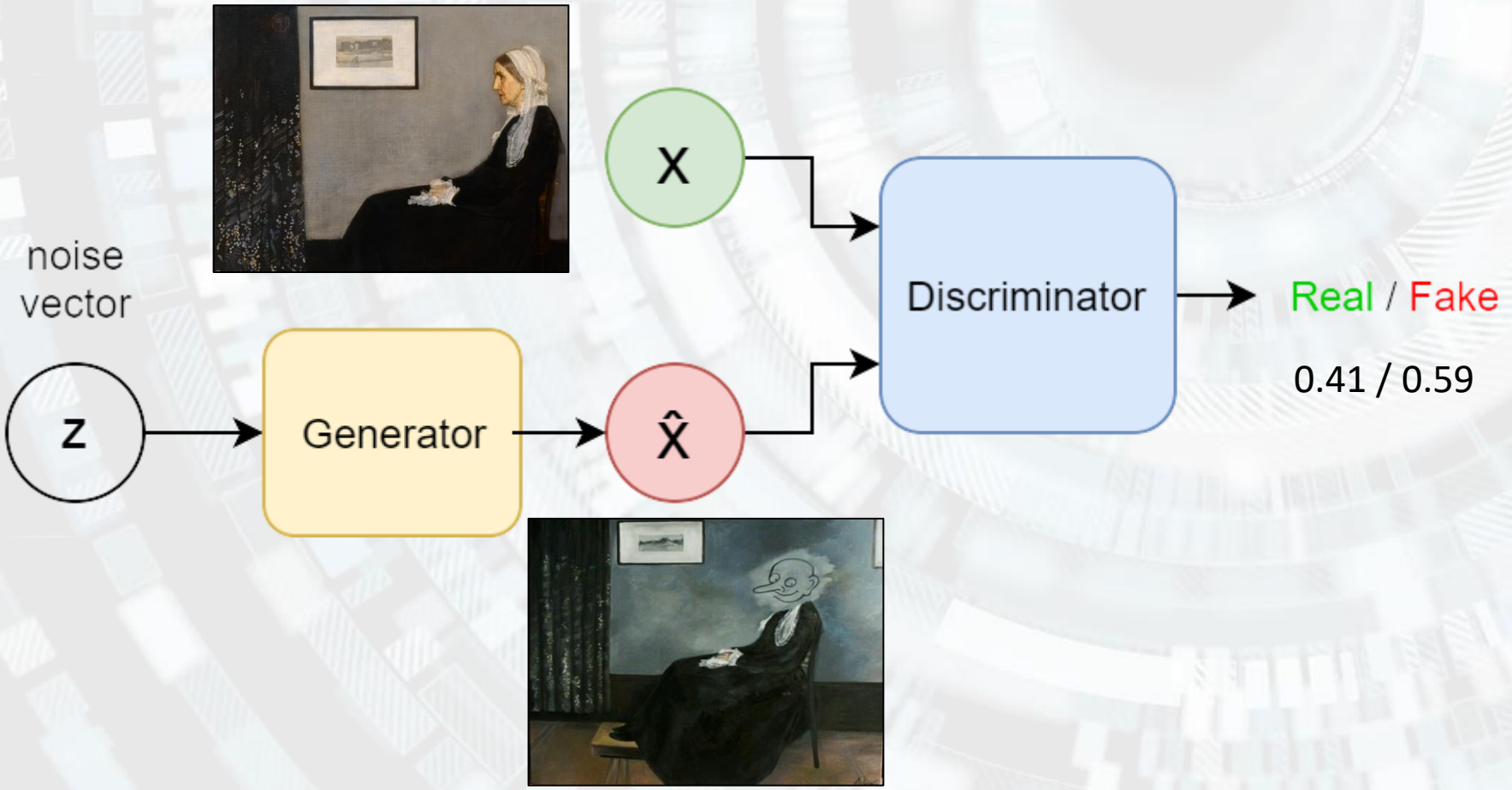
Generative Adversarial Networks



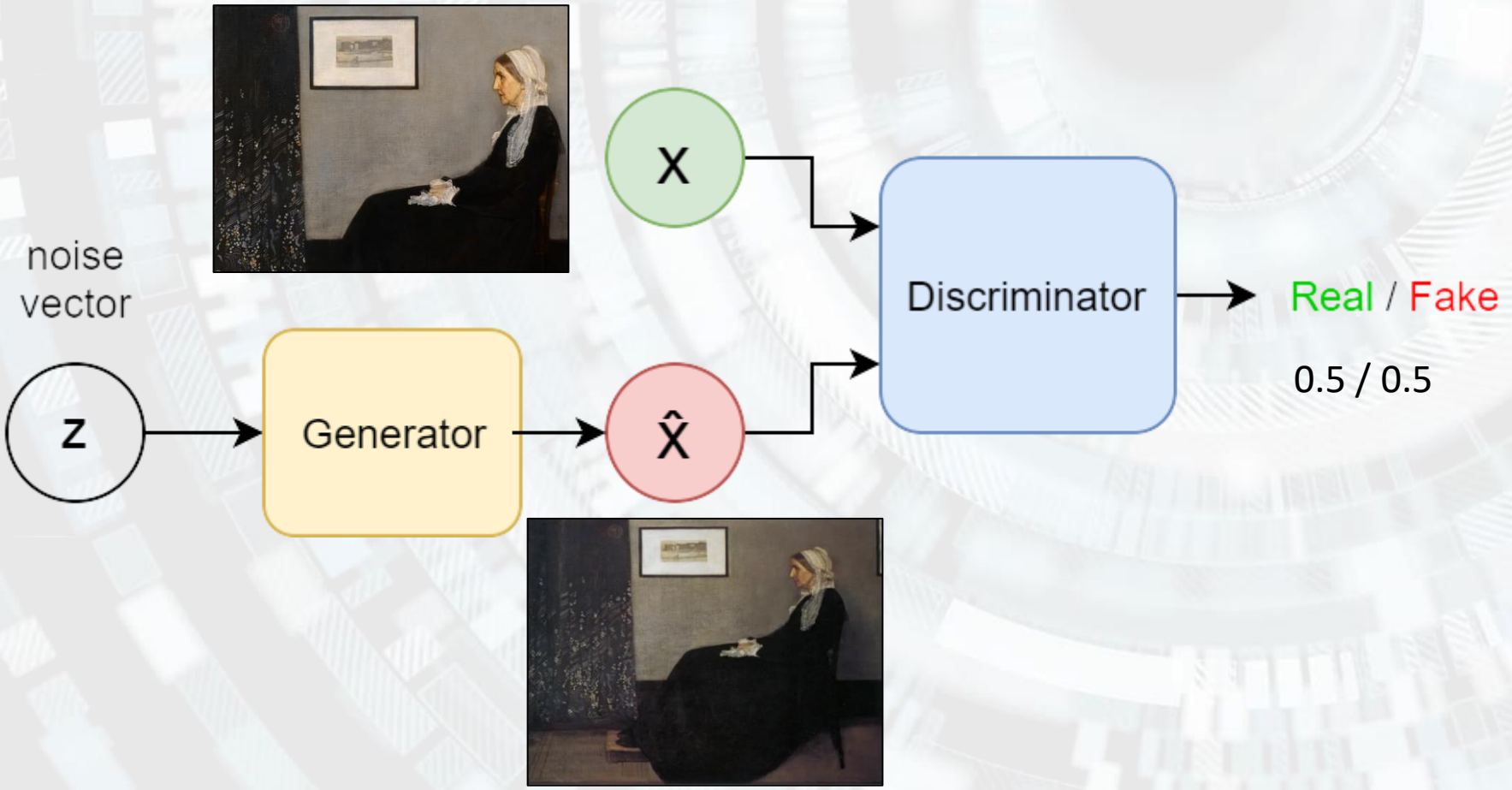
Generative Adversarial Networks



Generative Adversarial Networks



Generative Adversarial Networks



Generative Adversarial Networks

- GAN models shrink the Jensen-Shanon divergence between real data distribution and model distribution ($JSD(p_{data}||p_G)$).
- Convergence is attained when the discriminator confuses real samples with synthesized ones ($p_{data} = p_G$). In this case, $D(x) = \frac{1}{2}, \forall x$.
- Training a GAN involves reaching a Nash equilibrium – none of the parties involved (generator/discriminator) can improve its state by making changes only to its own parameters.

Generative Adversarial Networks

Limitations:

- Training is very unstable and convergence is hard to attain => mode collapse: the generator learns to generate a reduced set of samples that fool the discriminator.

Generative Adversarial Networks

Limitations:

- GANs need very large datasets for training (a data distribution as dense and uniform as possible). Otherwise, overfitting is obtained, similar to mode collapse.
- The evaluation of GANs depends on the task they perform. There is no standardized method to evaluate these models.

Generative Adversarial Networks

Limitations:

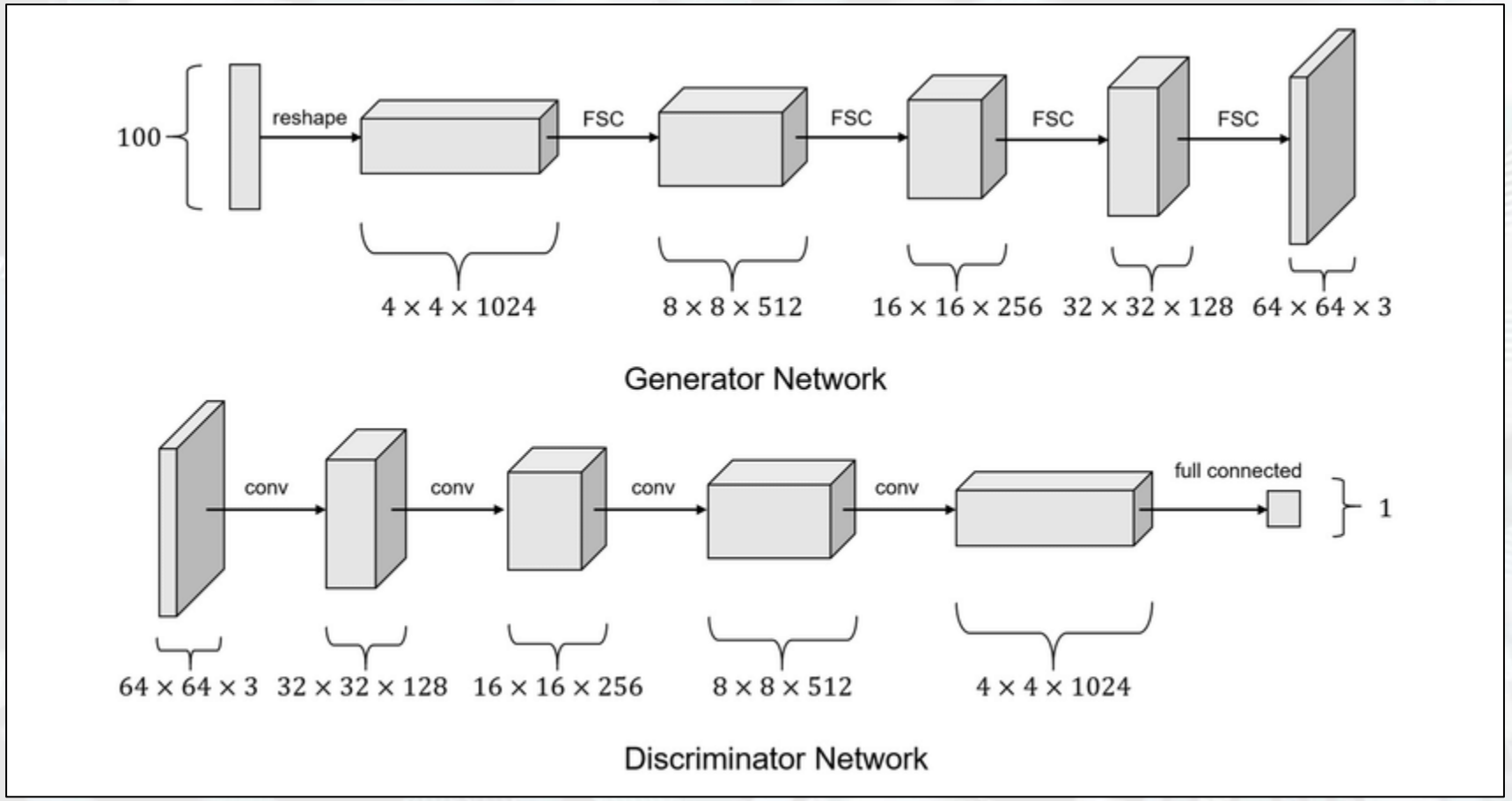
- Conceptually, GANs are trained to generate new sample starting from noise values => obtaining the noise vector that was used to generate a sample is hard to obtain.

Generative Adversarial Networks

Advantages:

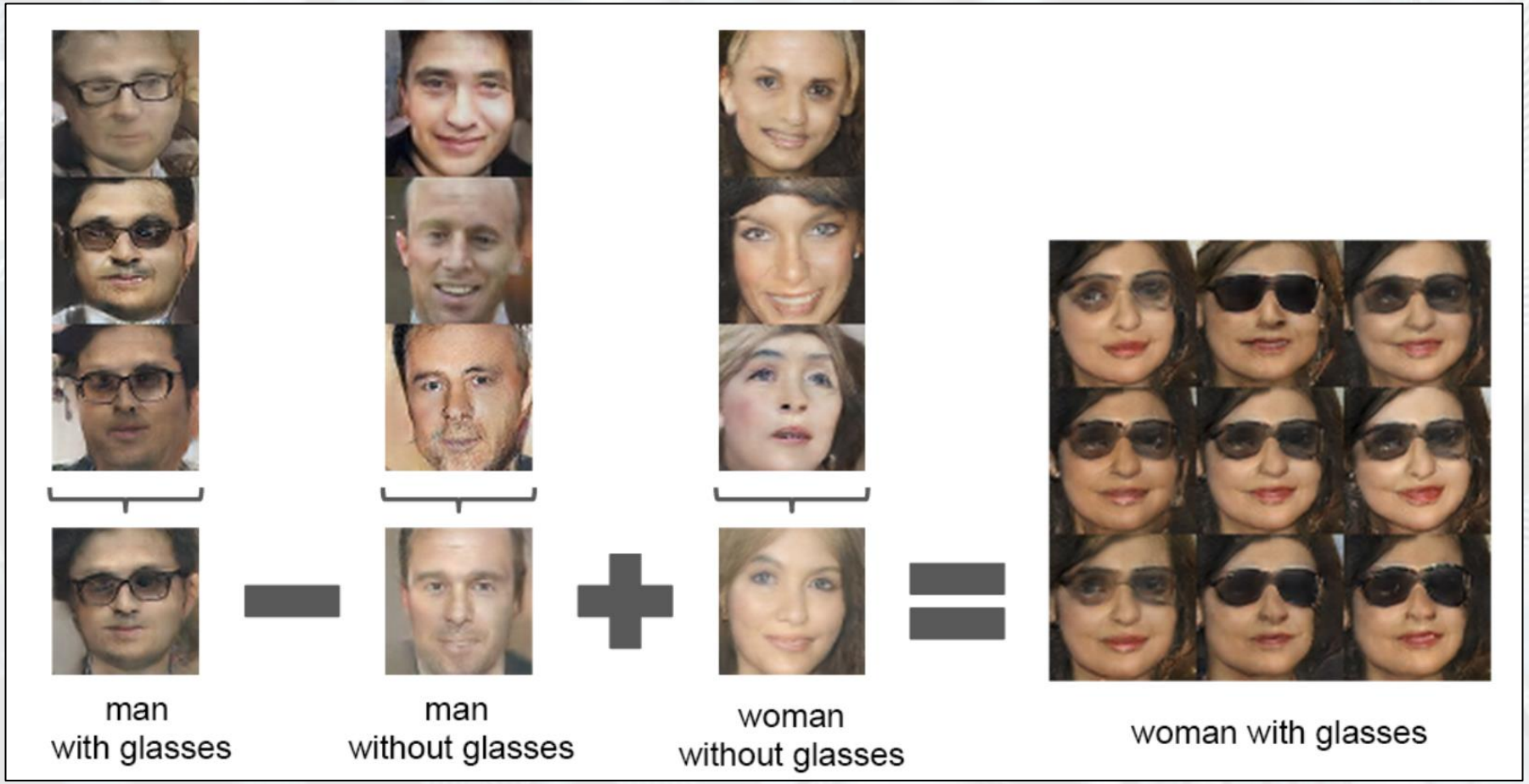
- Generates very clear samples;
- Fast inference (hundreds of samples/s can be generated);
- Easy to use for style transfer;
- Open applications in all fields (medical, automotive, artistic, engineering, education, etc.)
- Numerous open-source implementations available.

Generative Adversarial Networks – DCGAN



Generative Adversarial Networks – DCGAN

Generative Adversarial Networks – DCGAN



Generative Adversarial Networks – DCGAN

Let’s test it out – unit #10