Unsupervised Learning - Autoencoder

Rodrigo Serna Pérez

January 2022

comillas.edu

Today we'll talk about …

1. Unsupervised Learning
2. Autoencoders
3. Applications
4. Sparse Autoencoders
5. Variational Autoencoders

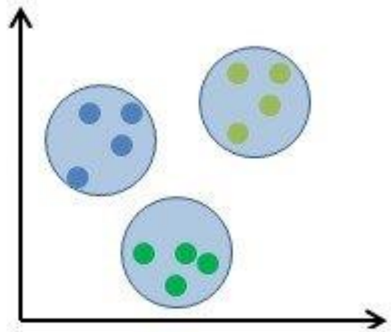
1

Unsupervised learning

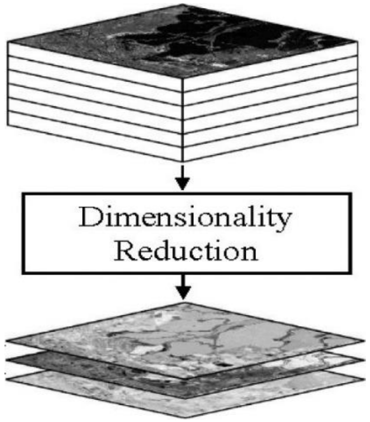
What is unsupervised learning?

- We have a huge dataset … but is not labeled. What can be do?

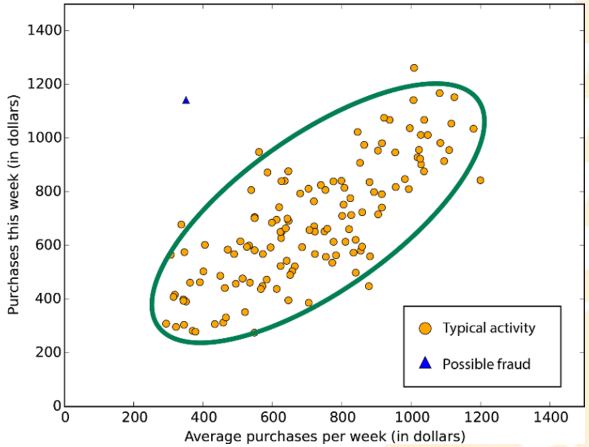
Clustering



Dimension Reduction



Anomaly Detection



Unsupervised Neural Networks?

- Neural Networks are supervised learning: if we do not have a target, we do not have an error, we do not have a loss, we do not have gradients …. no training
- Can we still solve unsupervised problems with deep learning?

2

Autoencoders

Autoencoders

- An autoencoder is a deep neural network that given an input tensor X tries to predict …
- The same tensor X
- Tries to emulate function $y = f(x)$ with $f(x) = x$
- Yes, you are right. We are training a network that does absolutely nothing.

Autoencoders

- Let's assume our input is a vector of size N.
- Let's split our dummy and useless network into two parts.
  - **Encoder**: a network with any size and any number of layers which expects a vector of size N and outputs a vector of size M being M < N (M outputs).
  - **Decoder**: a network with any size and any number of layers which expects a vector of size M and outputs a vector of size N (N outputs).
- Now we train both networks stacked together to predict X given X.
- We encoded a vector of size N into another of size M. **Dimensionality Reduction!!!!**

Encoder Decoder Intuition

- Imagine you want to send this message to a friend:

"autoencoders are very cool“

- But since you are a good millennial, you can't write all those letters. Instead, you will only write:

"autoncodrs r vry cool“

- You are dropping several letters (encoding), and your friend can still undestand the original message (decoder). **Together, you are an autoencoder!!**

Autoencoders and PCA

- **PCA:** when we are using PCA to convert a dataset $x \in \mathbb{R}^{DxN}$ into $y \in \mathbb{R}^{DxM}$, with $M < N$ we are computing two matrices $A \in \mathbb{R}^{NxD}$ and $B \in \mathbb{R}^{DxN}$ such as:

$$min \sum_i ||A * B * x_i - x_i||^2$$

So the vector x_i converts into Bx_i.

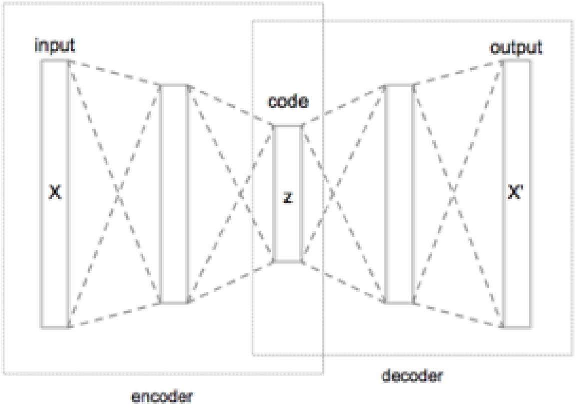
- In fact, **we can build PCA in keras** with a few lines of code.

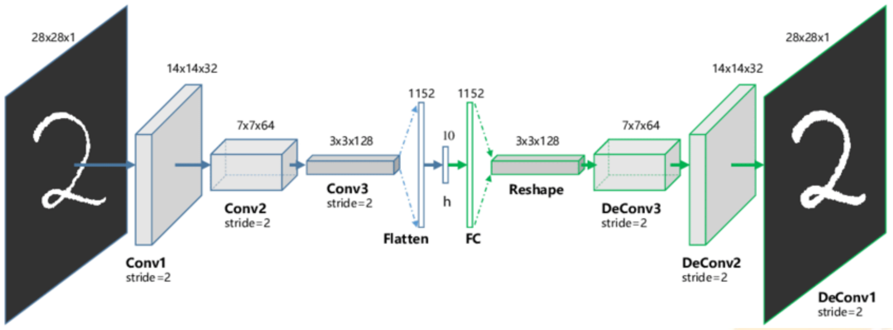
```
input_ = keras.layers.Input(shape=(N,))
x = keras.layers.Dense(M, activation="linear")(input_)
output = keras.layers.Dense(N, activation="linear")(x)
pca_model = keras.models.Model(input_, output)
pca_model.compile("adam", "mse")
pca.model.fit(X, X)
```

PCA = Linear Autoencoder

Deep Autoencoders

- PCA only captures linear dependencies, but **deep autoencoders can capture much more complex relations**!!
- We have freedom to choose the architectures for encoder and decoders (denses, CNNs …). We typically use **symmetrical architectures** for encoder and decoder.





Autoencoders

- **Intuition**: among the N initial variables there could be some correlations (linear and non linear).
- If one of the features is highly correlated from the others, it **could be dropped in the encoder and we can still get it back in the decoder**.
- The encoder will try to put similar input vectors into similar encoded vectors.

Implementation

```
[9]: INPUT_DIM = 64
     ENCODING_DIM = 10
     HIDDEN_LAYERS = [50, 30]
```

```
[20]: # Encoder
      input_layer = keras.layers.Input(shape=(INPUT_DIM, ))
      x = input_layer
      for i, h in enumerate(HIDDEN_LAYERS):
          x = keras.layers.Dense(h, activation="relu", name=f"encoding_hidden_{i}")(x)
      x = keras.layers.Dense(ENCODING_DIM, activation="sigmoid", name="final_encoding_layer")(x)
      encoder = keras.models.Model(input_layer, x, name="encoder")
```

```
[21]: # Decoder
      input_layer = keras.layers.Input(shape=(ENCODING_DIM, ))
      x = input_layer
      for i, h in enumerate(reversed(HIDDEN_LAYERS)):
          x = keras.layers.Dense(h, activation="relu", name=f"decoding_hidden_{i}")(x)
      x = keras.layers.Dense(INPUT_DIM, activation="tanh", name="final_decoding_layer")(x)
      decoder = keras.models.Model(input_layer, x, name="decoder")
```

```
[24]: # Autoencoder
      input_layer = keras.layers.Input(shape=(INPUT_DIM, ))
      encoding = encoder(input_layer)
      reconstruction = decoder(encoding)
      autoencoder = keras.models.Model(input_layer, reconstruction)
      autoencoder.compile("adam", "mse")
      autoencoder.summary()
```

```
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_15 (InputLayer)       [(None, 64)]              0

 encoder (Functional)        (None, 10)                5090

 decoder (Functional)        (None, 64)                5144

=================================================================
Total params: 10,234
Trainable params: 10,234
Non-trainable params: 0
_________________________________________________________________
```

```
[ ]: autoencoder.fit(X_train, X_train)
```

3

Applications

Dimension Reduction

- When we have a huge number of dimensions, we can use an autoencoder to **get a smaller vector**.
- Then we can **use the encodings to train another model** (Random forest, linear regression, other deep learning algorithm …).

Outlier/ Anomaly Detection

- We can use our trained autoencoder to **detect outliers and anomalies**.
- **Anomaly**: data which does not seem to belong to the same distribution than the others.
  - Example: having as features the age of a person and the money in their bank account, we expect very low money when the person is under 10 years old. If there is a 5 years old boy with 10 millions in his account, it is an anomaly.
- The autoencoder would not be able to learn how to encode and decode those “weird events”. **We expect a higher reconstruction loss when there is an anomaly**.

Noise reduction

- What happens if our inputs are corrupted by random noise?

$$X_i^{corrupted} = X_i + n$$

- The autoencoder can learn how to encode and reconstruct the signal, but **never the noise**.
- When we pass $X_i^{corrupted}$ through the autoencoder, **we will get a new signal with reduced noise!!**
- **Denoising Autoencoder**: when we have access to signals with no noise, we can add random noise and build a network that tries to predict X_i from $X_i + n$.

Semi Supervised Learning

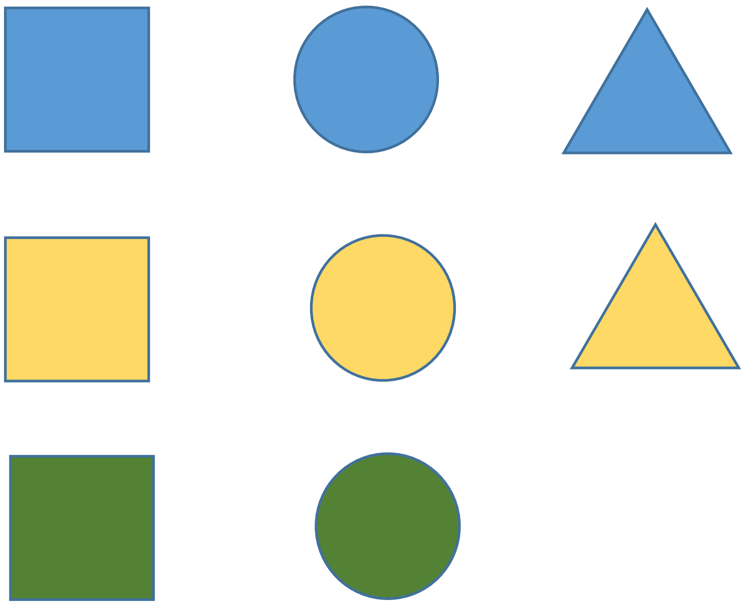
- Semi Supervised Learning refers to problems in which we have **access to both labeled and not labeled datasets**.
- How can be used all available data?
- We can use all the not labeled data to **build an autoencoder**.
- Using the encoder, we **get the encoding representation for the X features** in the labeled dataset.
- We can **build a supervised regression/classification algorithm** using the encodings of X and the labels.
- **This approach is very powerful** in language related tasks.

3

Sparse Autoencoders

Autoencoders and overfitting

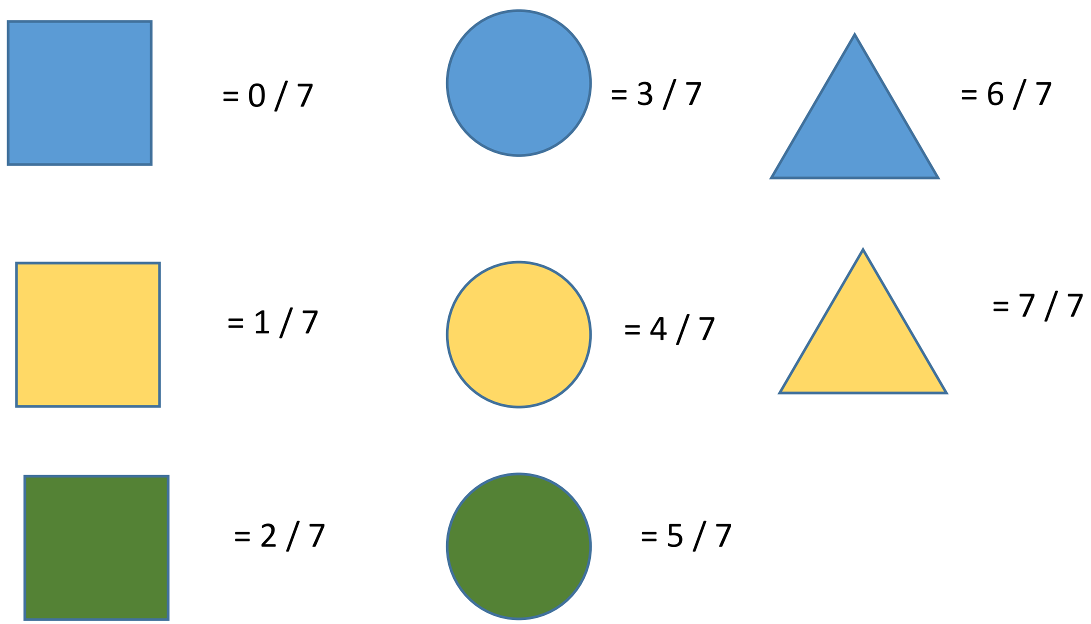
- Could a human solve the autoencoder problem manually?



We have these 8 different samples. How could we encode them?

Autoencoders and overfitting

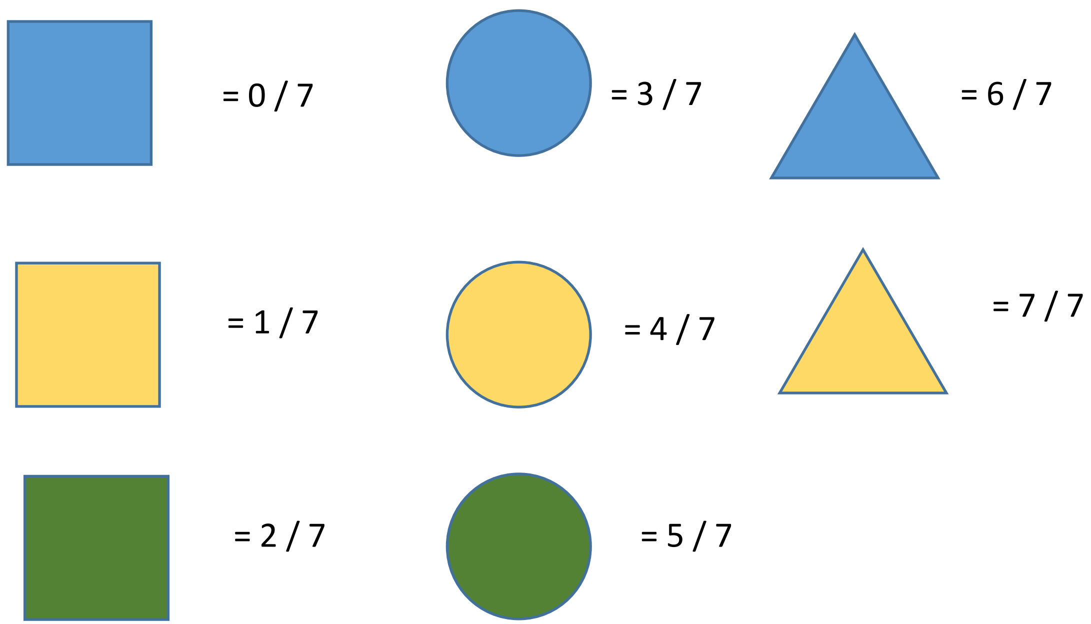
- **Solution 1**: encoding with no error and an embedding size of 1.



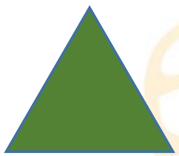
Problem Solved!!!

Autoencoders and overfitting

- **Solution 1**: encoding with no error and an embedding size of 1.



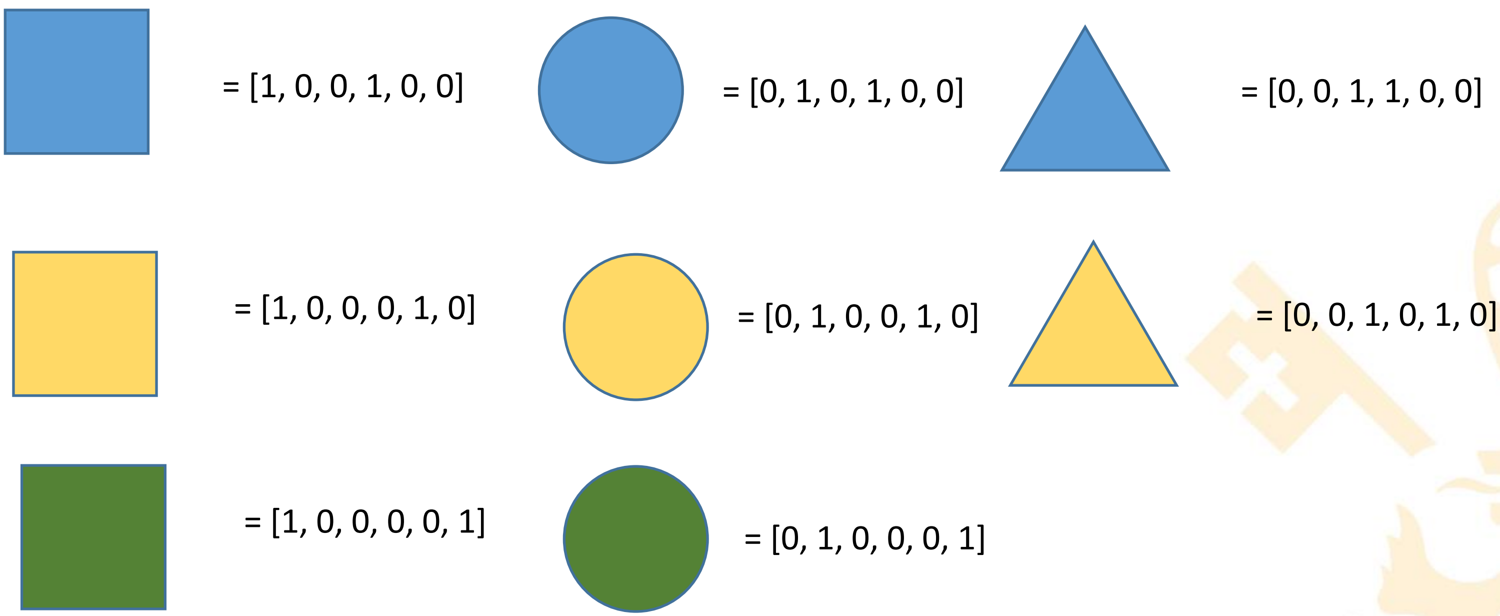
Now we received a different sample which we had not seen



Ups … there is no way to encode it.
We were overfitting!!!

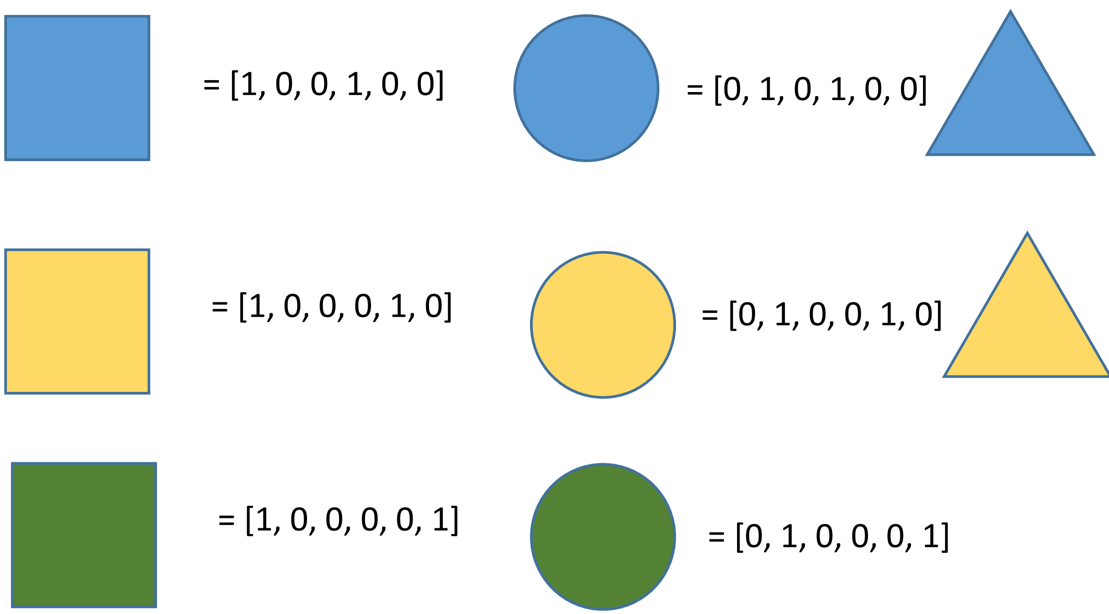
Autoencoders and overfitting

- **Solution 2**: better encoding with dimension of 6.
  Three first values will encode the shape, three last will encode the colour.



Autoencoders and overfitting

- **Solution 2**: better encoding with dimension of 6.
  Three first values will encode the shape, three last will encode the colour.



Now we received a different sample which we had not seen

We can encode it as:

[0, 0, 1, 0, 0, 1]

Our model can generalize!!

Sparse Autoencoder

- We can **encourage the sparsity of the values in the latent space** in order to avoid the overfitting. This is what we call the Sparse Autoencoder.
- We do it through **Regularization**. We modify our loss error to:

$$loss = loss_{reconstruction} + \lambda \ f(encoding \ values)$$

Being f(x) any function that encourages sparsity (typically l1 norm)

- In sparse autoencoders, **using a higher dimension in the embedding than in the input makes sense**!!

Regularizer in keras

- **Keras have built-in utilities** to build models with regularization.

```
from tensorflow.keras import regularizers
from tensorflow import keras

dense_layer = keras.layers.Dense(10, activation="tanh", activity_regularizer=regularizers.l1(0.01))
```

- Now, all models that use the layer will use l1 regularization.

4

Variational Autoencoders

Generative Models

- **Generative Models** learn from a given dataset $\{X_1, X_2, \dots, X_N\}$ how to produce new samples that follow the same distribution.
- It needs to **learn the distribution f(x)** of the data.
- Example: given a set of pictures of faces, generate new pictures of faces.

Montecarlo method

- There is a simple method that can generate new samples from a density distribution f(x): **Montecarlo method**.
- We take a random sample p from an uniform distribution and the generated value p is the one at which F(x) = p

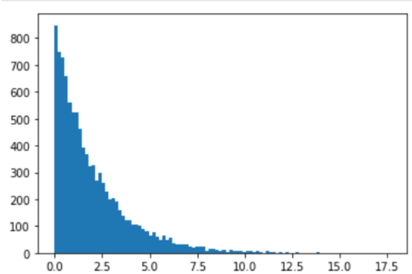
```python
[44]: # GENERATES VALUES THAT FOLLOW AN EXPONENTIAL DISTRIBUTION

lambda_ = 0.5

# inverse function of distribution function of exponential F(t) = 1 - exp(-lambda t)
inverse_function = lambda p: - np.log((1 - p) )/ lambda_

# generate values randomly in [0, 1]
uniform_values = [np.random.uniform() for _ in range(10000)]

# The generated values are the one at which F(t) = p, for p coming from the uniform distribution
generated_values = [inverse_function(v) for v in uniform_values]
plt.hist(values, bins=100);
```



Faces generation using this method is not viable, but let's keep the idea of generating random variables from other random variables.

Latent Variable

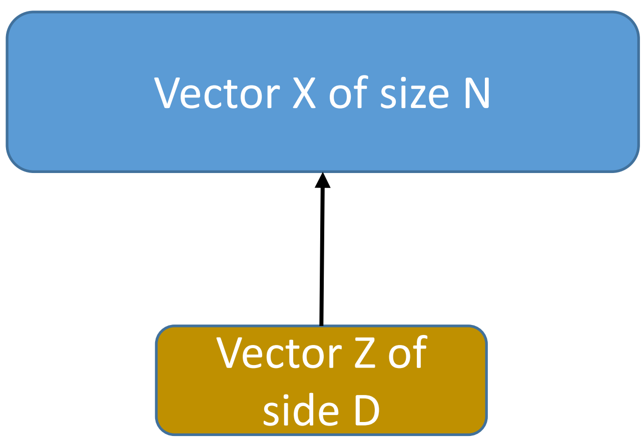
- **Hypothesis**: a random variable $X \in \mathbb{R}^N$ depends on other hidden (latent) variable $Z \in \mathbb{R}^D$ where $D \ll N$.
- In fact:

$$f(x) = \int f({}^{x}/_{z}) f(z)\ dz$$

- Example: a 100 x 100 image of a face will contain 10,000 variables (one per pixel). But those variable depend on a few random latent variables such as sex, race, age, how much smiling …
- The **hidden space Z of D random variables** is precisely from where we will sample to generate our outputs (remember Montecarlo!!!)

Generating from hidden samples

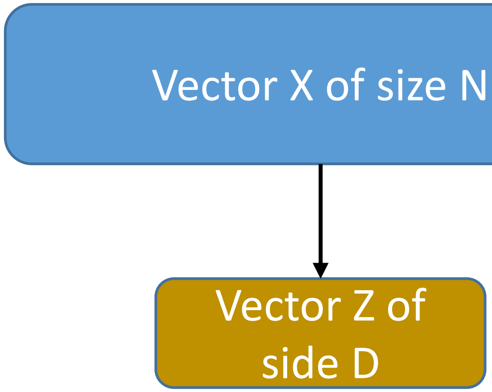
- Face generation problem: assume we have manually labeled D features (sex, race, smiling …) for each face. We could build a neural network that generates an image given a vector of features.



Yes, it is a decoder !!

Hidden features generation

- Ok, but which features must be select? And … who is willing to spend a week labeling thousands of images of faces?
- Let a Neural Network do it !!!
- Given an image of size N, **the network must generate D latent variables.**



OK, that is an encoder.

BUT WAIT!!!

Hidden features generation

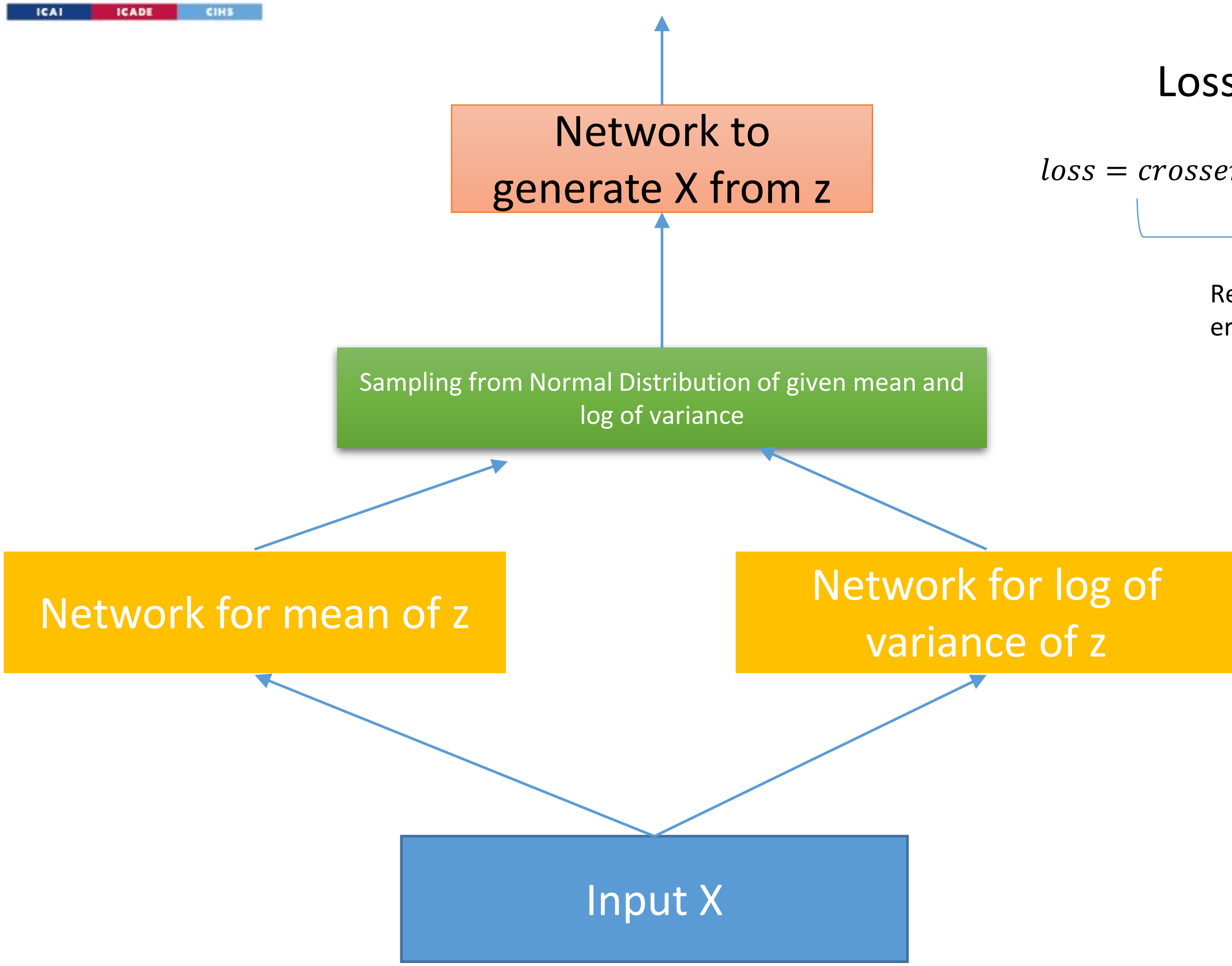
- Remember, later we want to **randomly sample from the latent space**. We must enforce the latent space to be a random variable that follows a known random distribution!!!
- We typically want the Z variable to follow a standard D-dimensional normal distribution. So:

$$z \sim N(0, \mathrm{I})$$

- We will build two different networks, **one for the mean and one for variance** (in practice it will be the logarithm of variance, later we'll see why).
- When trained, we will **use the model to encode the input X as a distribution:**

$$z \sim N(\mu(X), \Sigma(X))$$

Variational Autoencoder



Loss function

$$loss = \underbrace{crossentropy(y, x)}_{\text{Reconstruction error}} + \underbrace{f(mean_z, logvar_z)}_{\text{Regularization, encourages z to be a standard normal distribution}}$$

Kullback–Leibler divergence

- The Kullback-Leibler divergence computes **how different are two distributions.**
- Given two distributions p(x), q(x), the Kullback-Leibler divergence is given by:

$$KL\big(p(x), q(x)\big) = \int p(x) * \ln\left(\frac{q(x)}{p(x)}\right) dx$$

- When p follows a multivariate diagonal normal distribution and q follows a standard normal distribution is:

$$KL\left(N\big(\mu, \Sigma_{diagonal}\big), N(0, I)\right) = \frac{1}{2} * \sum_{i=1}^{k} \left(\sigma_i^2 + \mu_i^2 - 1 - \ln\big(\sigma_i^2\big)\right)$$

- This is the **expression that will be used as regularization**.

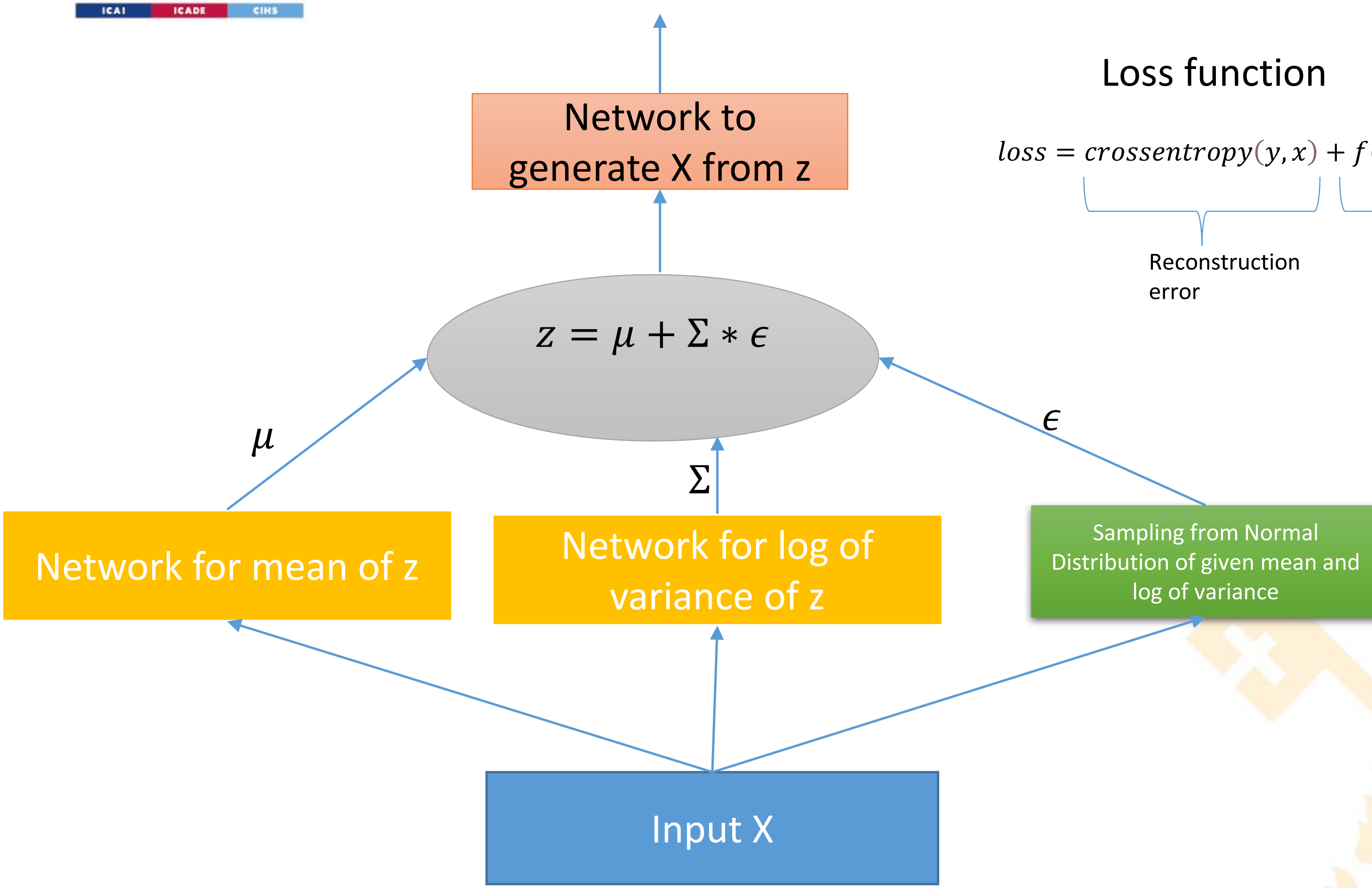
Training Variational Autoencoder

- So the complete expression for the loss is:

$$loss = binary\ crossentropy(y, x) + \frac{1}{2} * \sum_{i=1}^{k} (\sigma_i^2 + \mu_i^2 - 1 - \ln(\sigma_i^2))$$

- Both terms are differentiable with the weights of the network, so gradient descent can be used. But **there is a bottleneck**!!
- The crossentropy term cannot be differentiable with the weights in the encoder because there is a sampling.
- **Reparametrization trick**: instead of sampling from $z \sim N(\mu, \Sigma)$, we sample $\epsilon \sim N(0, \mathrm{I})$ and then $z = \mu + \Sigma * \epsilon$.
- Using this trick, the reconstruction error can be backpropagated.

Variational Autoencoder – Reparameterization Trick



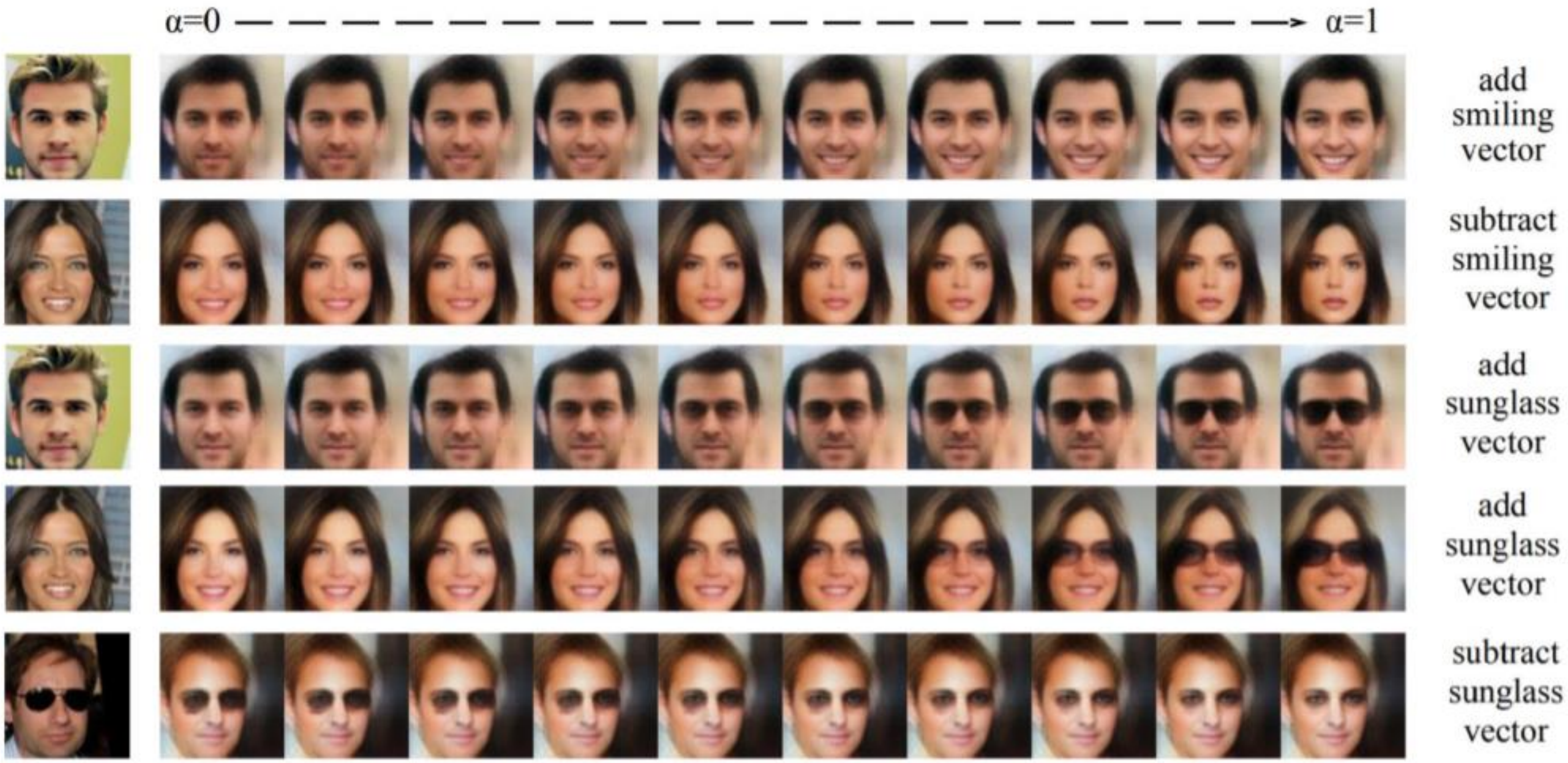
Loss function

$$\underbrace{loss = crossentropy(y,x)}_{\text{Reconstruction error}} + \underbrace{f(mean_z, logvar_z)}_{\text{Regularization, encourages z to be a diagonal normal distribution}}$$

Latent Space properties

- If z_1 and z_2 are close, they will generate similar samples $x_1$and x_2
- We can perform arithmetical operations in the latent space.
- Example: if we take the average z for all faces that are smiling and subtract the average z for all faces that are not, we get the “add smile vector”.
- Then, the sum of a “not smiling face” + “add smile vector” is the “smiling face” picture.

Latent Space



Current Developments for generative models

- Currently, variational autoencoders have been outperformed by other type of models:
  - Autoregressive models
  - Generative Adversarial Networks (GANs)
- VANs have the advantage of easy access to the latent space ...
- But the output is usually “blurred”.
- Several modifications have been proposed to get better results and is a current line for research.

Figure 1: 256×256-pixel samples generated by NVAE, trained on CelebA HQ [28].

Paper for NVAE (2020)

Data Augmentation

- VAEs are very useful when we don’t have enough data to train a model or classes are extremely imbalanced.
- Using a generative model, we can **generate “fake data”** and use it as training data.
- **Example:** classification problem with 99.99% of negative samples and 0,01% of positive samples.
  - Step 1: get the positive samples and train a generative model
  - Step 2: generate new data using the generative model till number of positives and number of negatives are in the same order.
  - Step 3: train a supervised classification algorithm using the augmented dataset that mixes real and fake data.
  - Step 4: evaluate only on real data!! Don’t use fake data for evaluating!!

COMILLAS

UNIVERSIDAD PONTIFICIA

ICAI ICADE CIHS

comillas.edu