Deep Learning

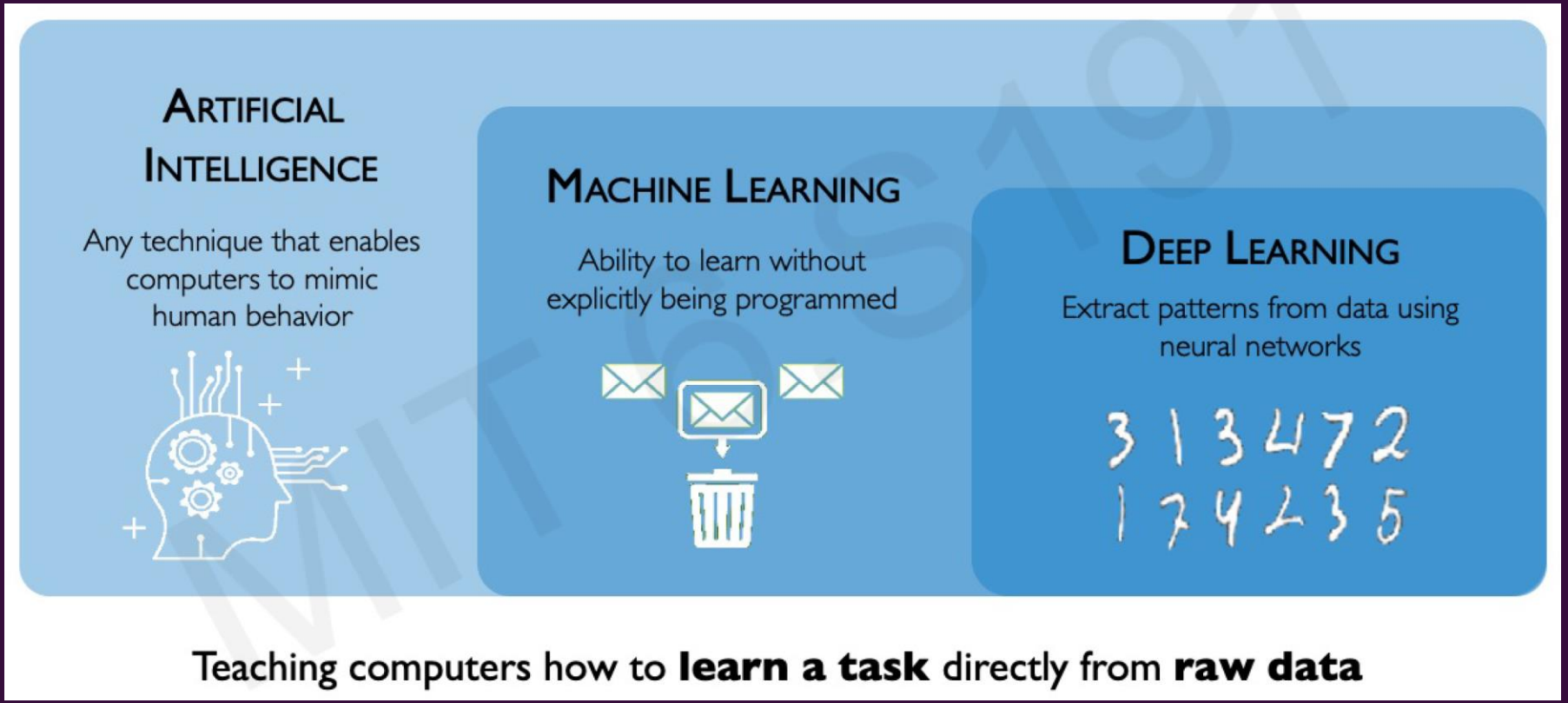
Outline:

- **Neural networks**
- **Convolutional Neural Networks**
- **Deep Generative Learning**

Co-funded by the European Union

What is Deep Learning?

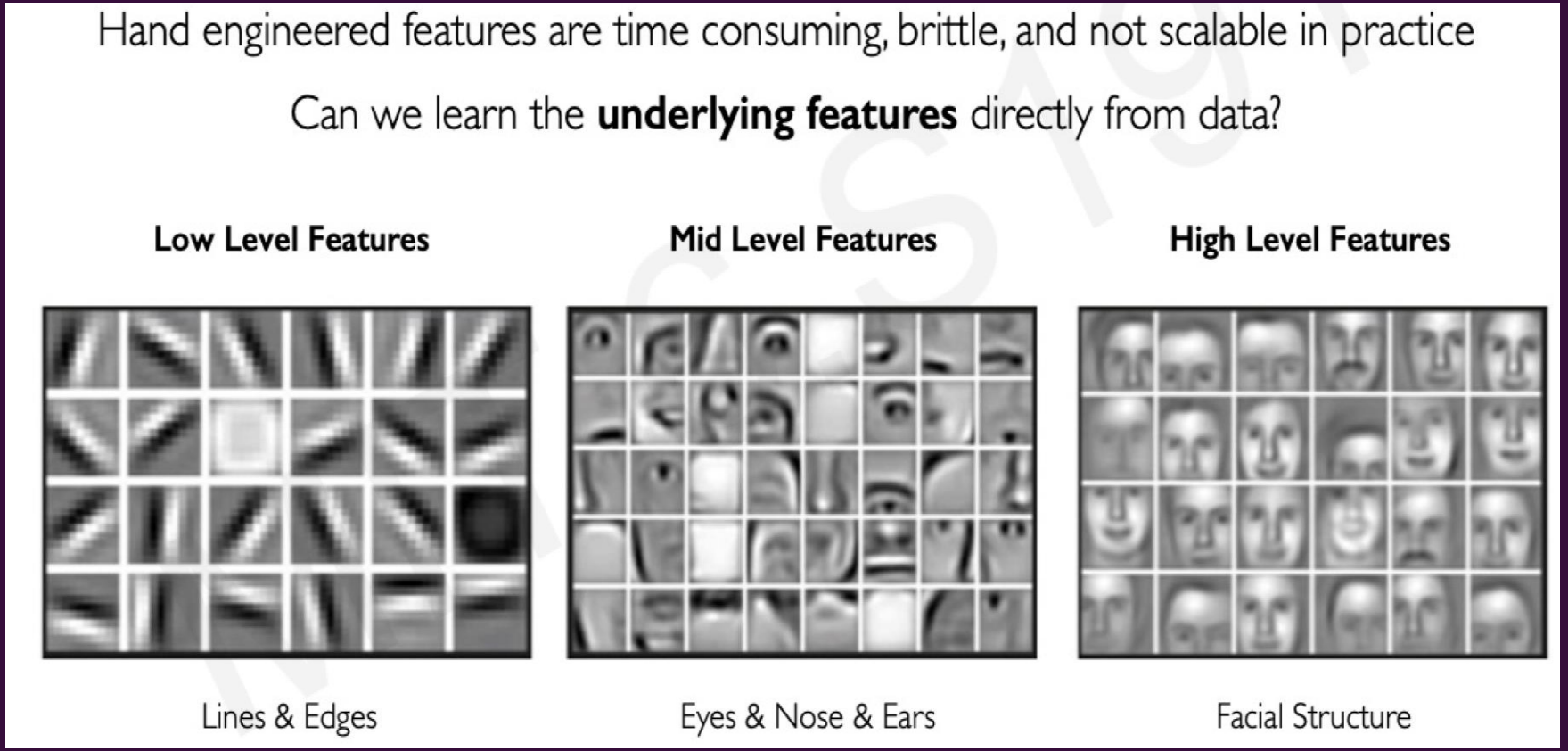
The hierarchical relationship shows how deep learning builds upon previous AI advances while adding its own unique capabilities



Co-funded by the European Union

Why Deep Learning?

- Deep learning automatically learns relevant features at multiple levels of abstraction
- Lower-level features combine to form increasingly complex and meaningful representations

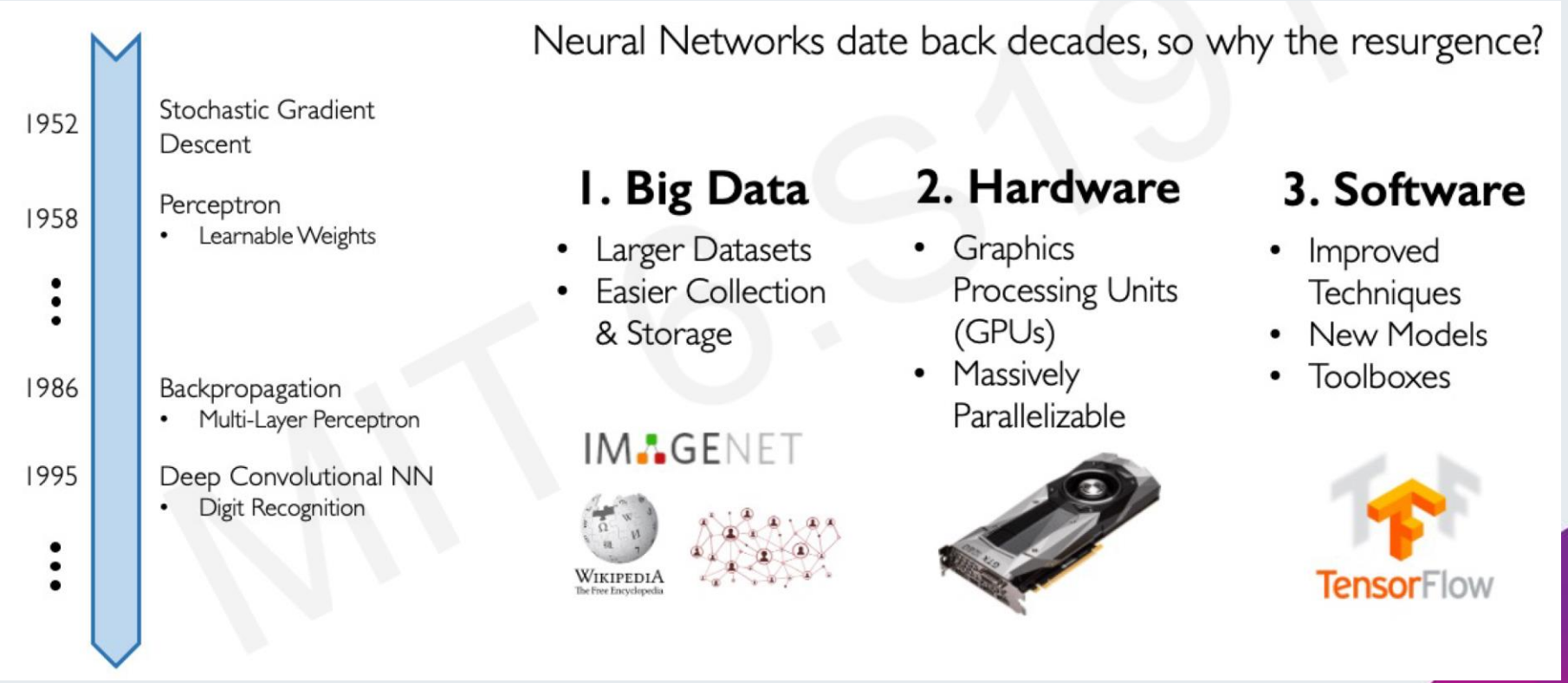


Co-funded by the European Union

Why Now?

- The field of neural networks has existed since the 1950s but only recently became practically viable

- Three key developments enabled modern deep learning: availability of big data, powerful GPU hardware, and improved software tools



Co-funded by the European Union

The Perceptron

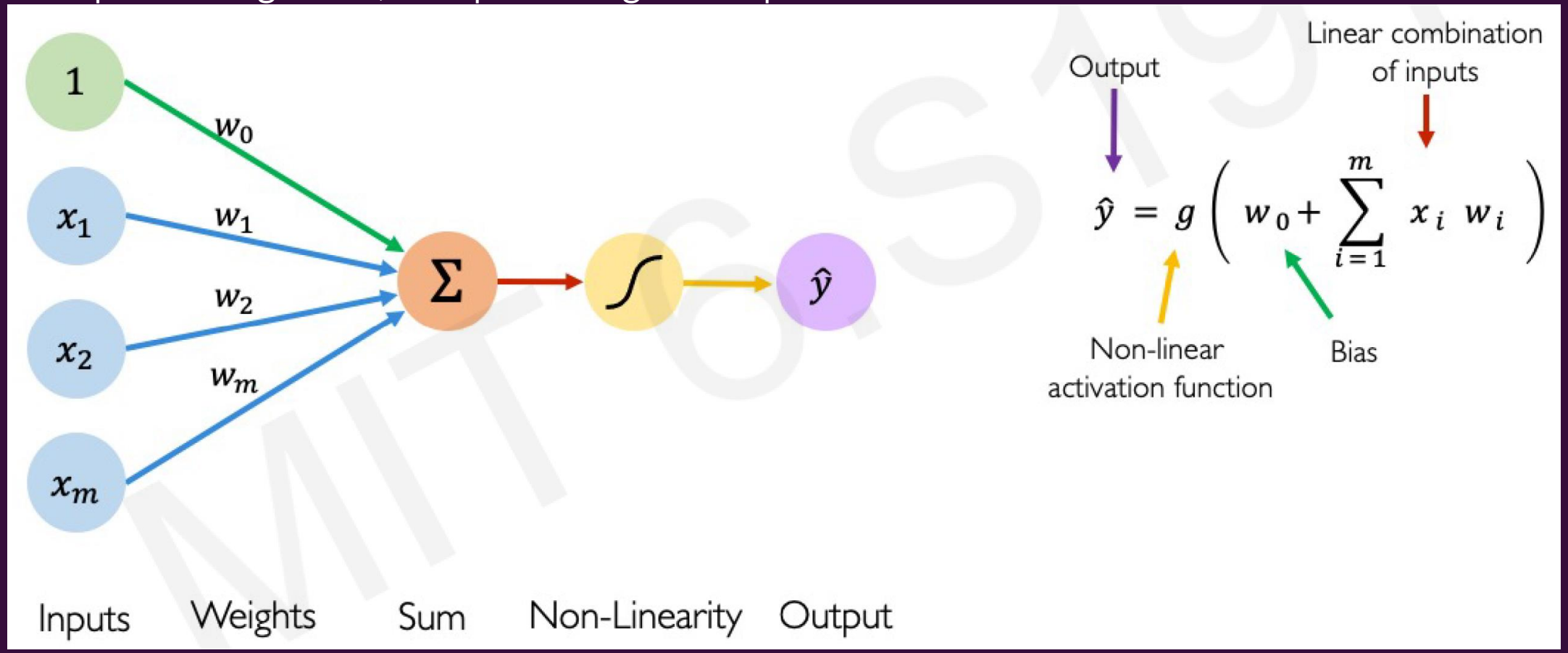
The structural building block of deep learning

Digital 4 Business
Evolving your digital future

Co-funded by
the European Union

The Perceptron: Forward Propagation

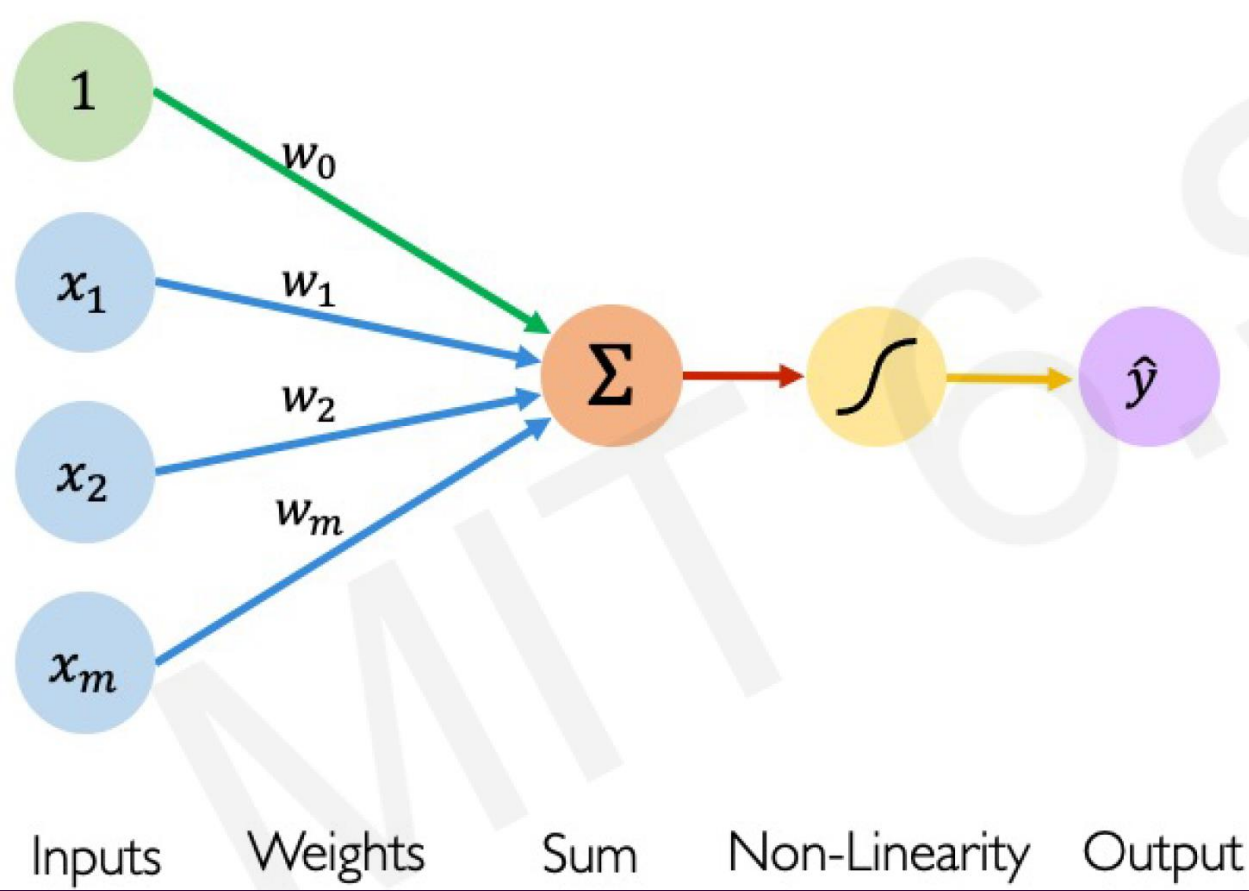
Think of it as a mathematical model of a biological neuron - taking inputs, processing them, and producing an output



Co-funded by the European Union

The Perceptron: Forward Propagation

- Each perceptron computes a weighted sum of its inputs plus a bias term



$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i\, w_i\right)$$

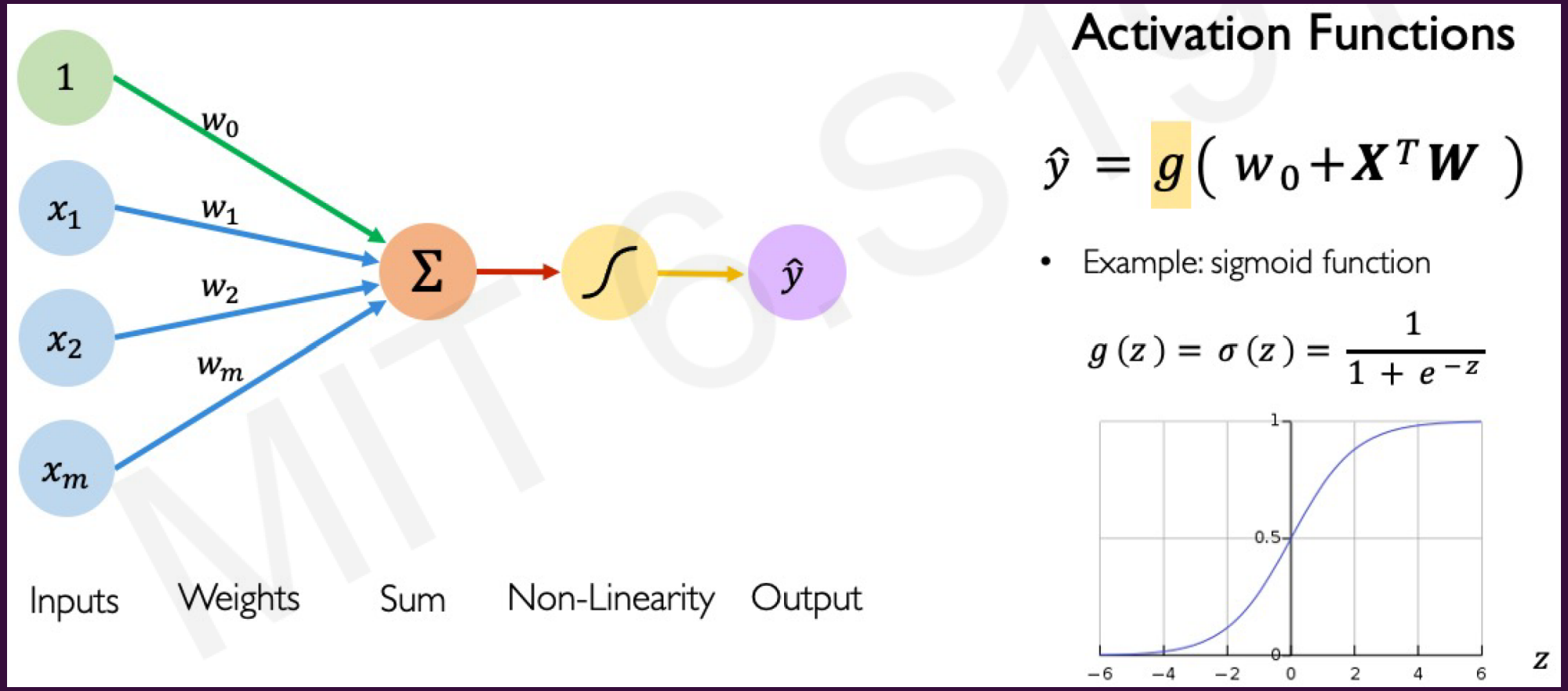
$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W}\right)$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

Co-funded by the European Union

The Perceptron: Forward Propagation

- The sum passes through an activation function that introduces non-linearity into the network

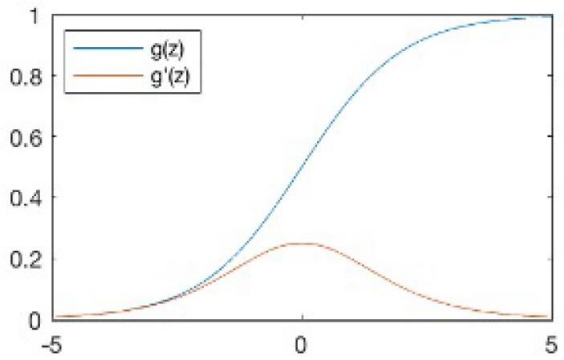


Co-funded by the European Union

Common Activation Functions

Activation functions determine whether and how strongly a perceptron "fires" based on its inputs

Sigmoid Function

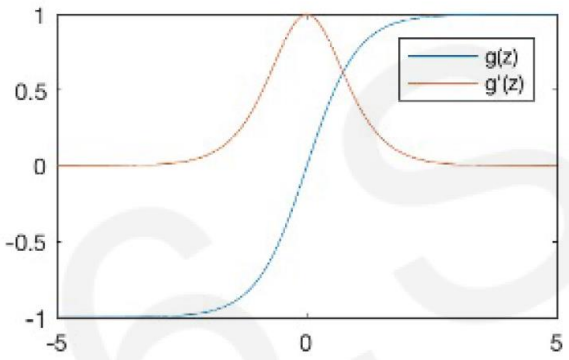


$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

`tf.math.sigmoid(z)`

Hyperbolic Tangent

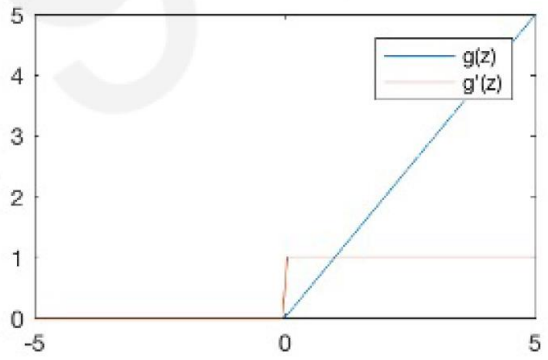


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

`tf.math.tanh(z)`

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

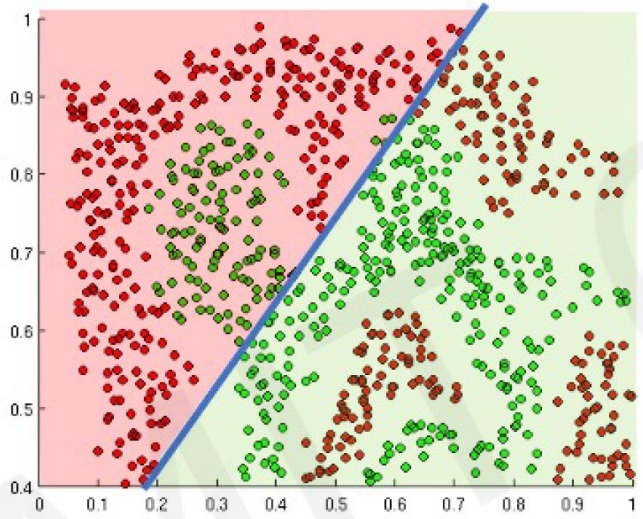
$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

`tf.nn.relu(z)`

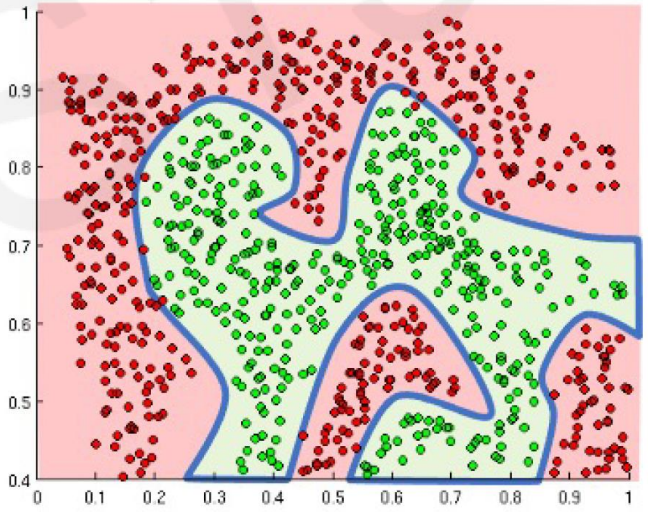
Co-funded by the European Union

Importance of Activation Functions

*The purpose of activation functions is to **introduce non-linearities** into the network*



Linear activation functions produce linear decisions no matter the network size

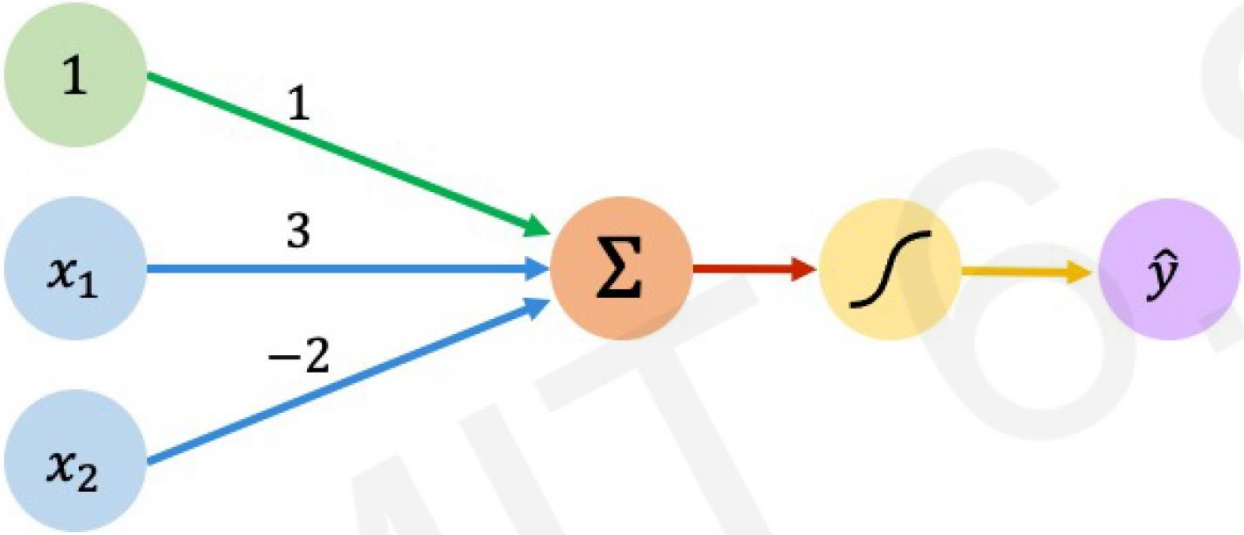


Non-linearities allow us to approximate arbitrarily complex functions

Co-funded by the European Union

The Perceptron: Example

- In this practical example, we have two input features (x_1, x_2) and specific weights (3, -2)
- The bias term ($w_0 = 1$) helps shift the decision boundary without changing its orientation



We have: $w_0 = 1$ and $\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$

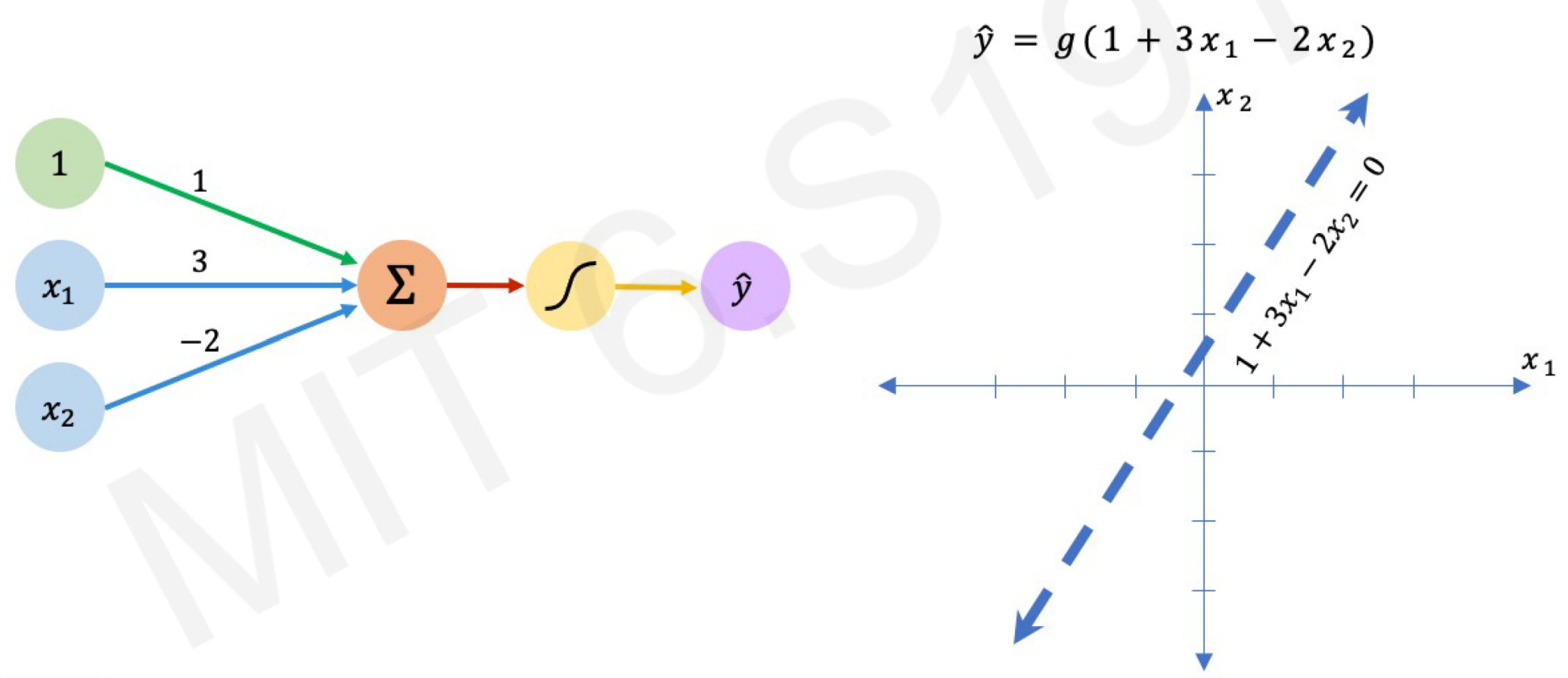
$$\hat{y} = g(\underbrace{1 + 3x_1 - 2x_2}_{})$$

This is just a line in 2D!

Co-funded by the European Union

The Perceptron: Example

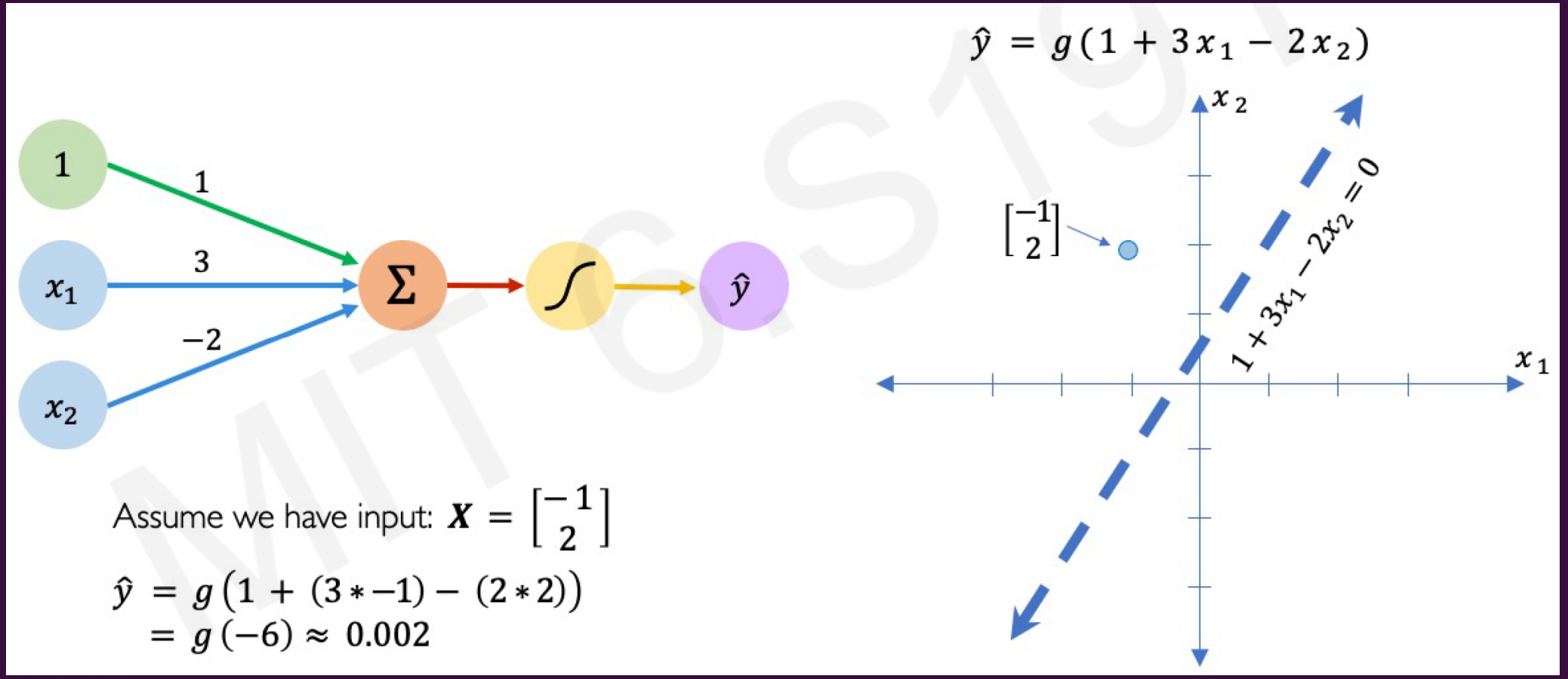
When mapped in 2D space, the perceptron creates a linear decision boundary



Co-funded by the European Union

The Perceptron: Example

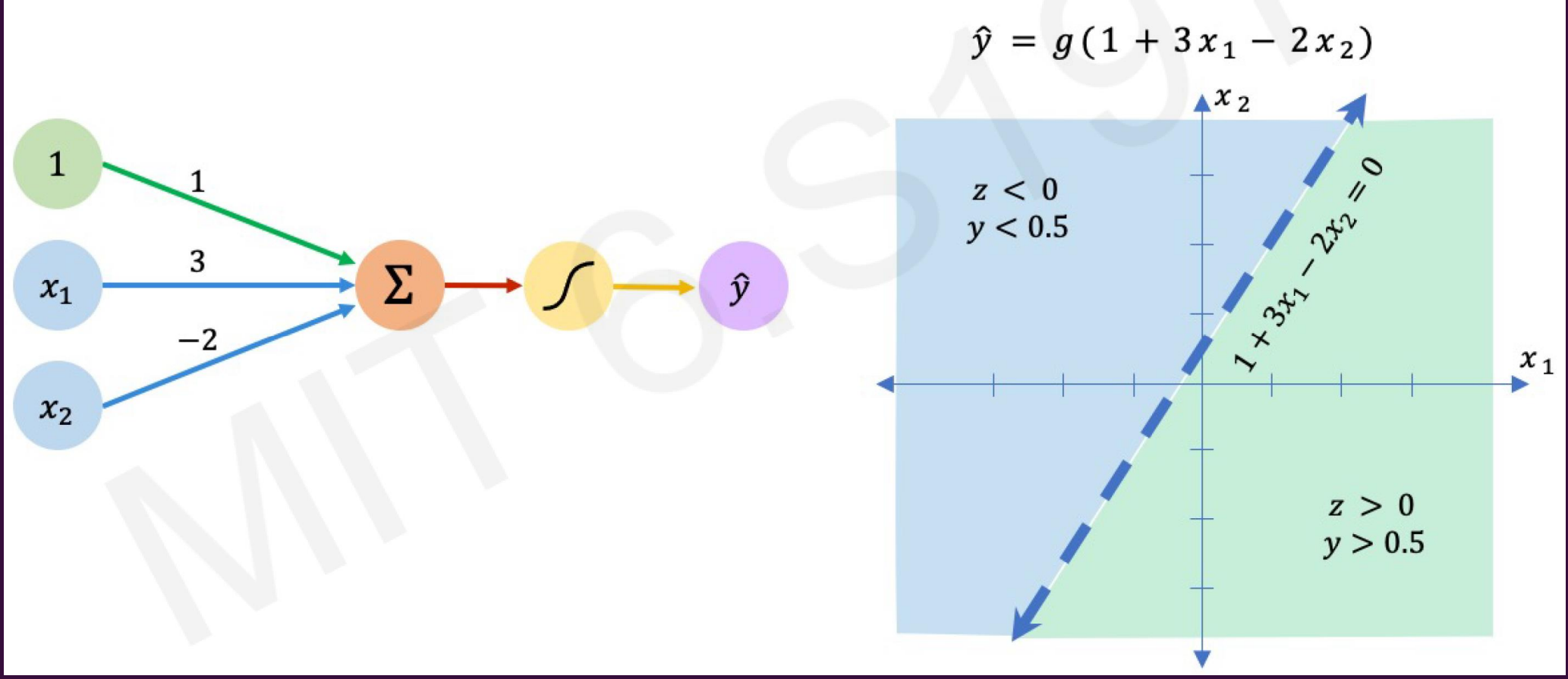
Understanding how numbers flow through this simple example helps grasp more complex networks



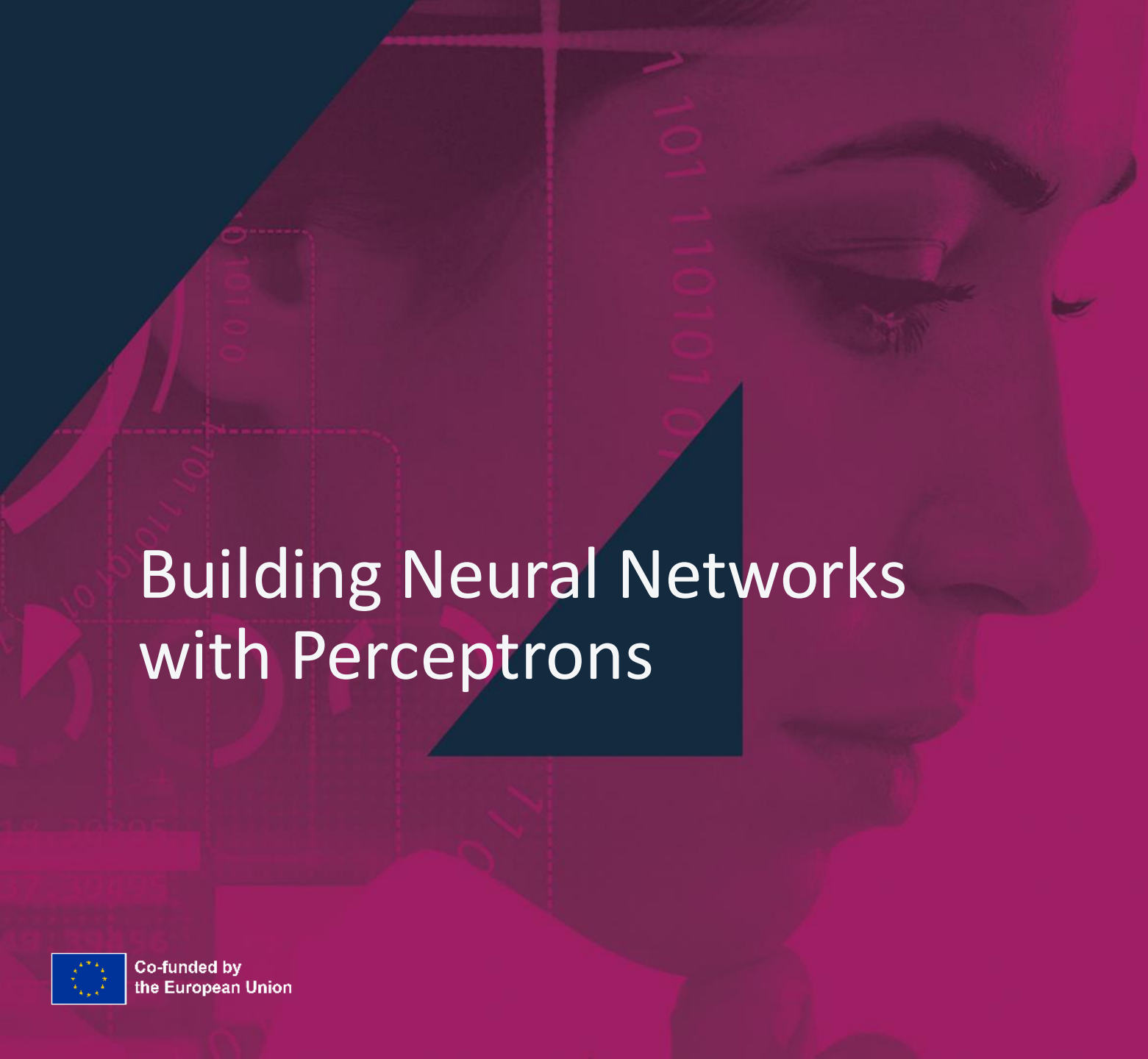
Co-funded by the European Union

The Perceptron: Example

Understanding how numbers flow through this simple example helps grasp more complex networks



Co-funded by the European Union

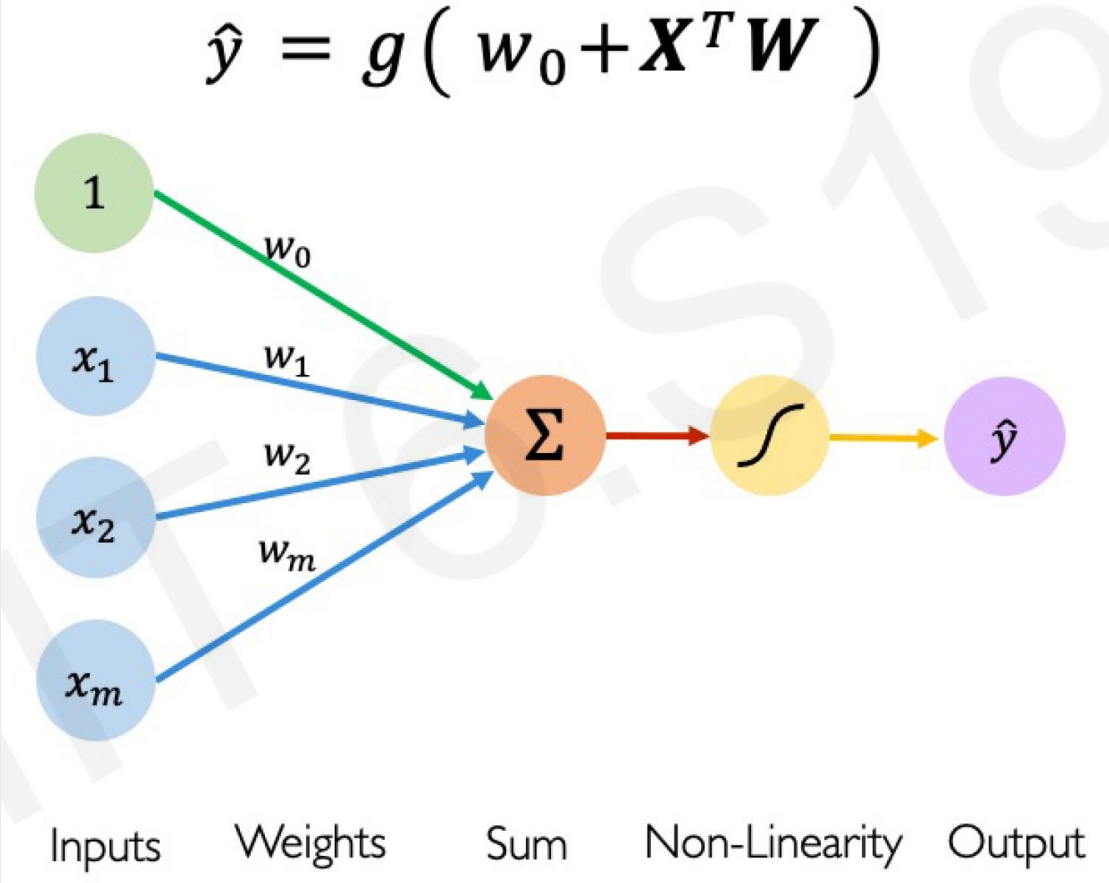


Building Neural Networks with Perceptrons

Digital 4 Business
Evolving your digital future

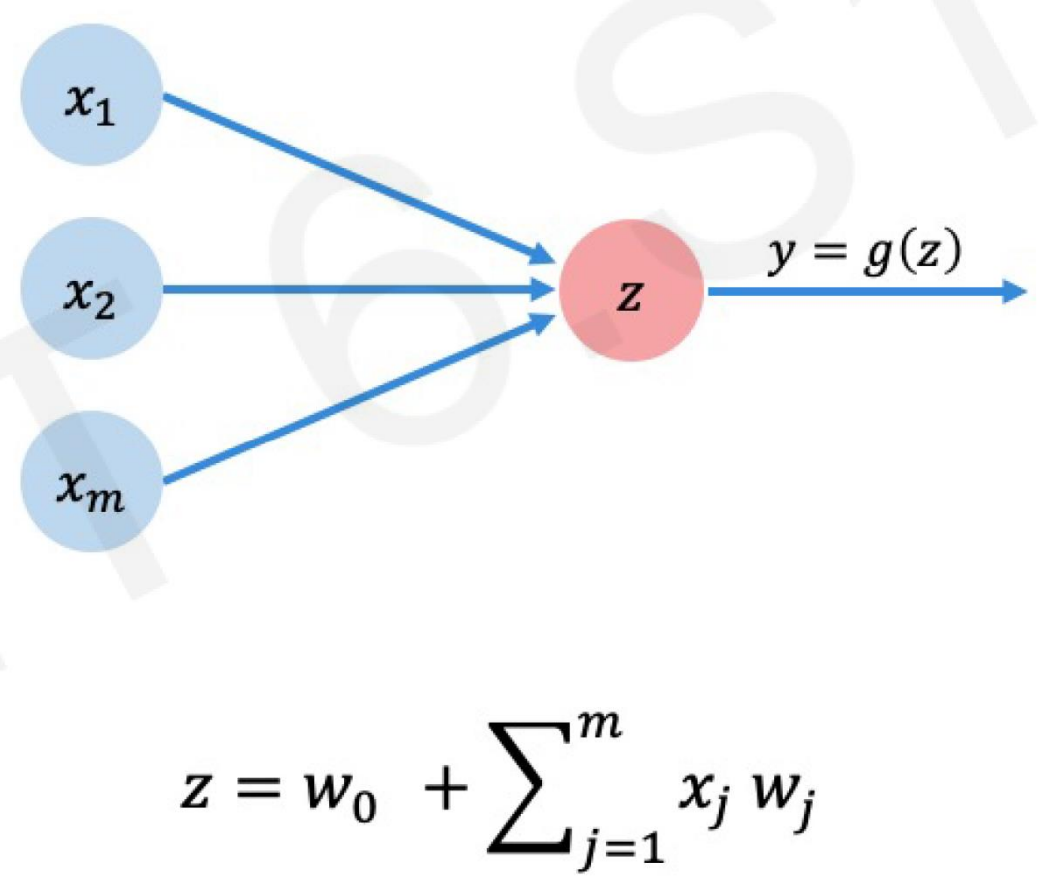
Co-funded by the European Union

The Perceptron Simplified



Co-funded by the European Union

The Perceptron Simplified

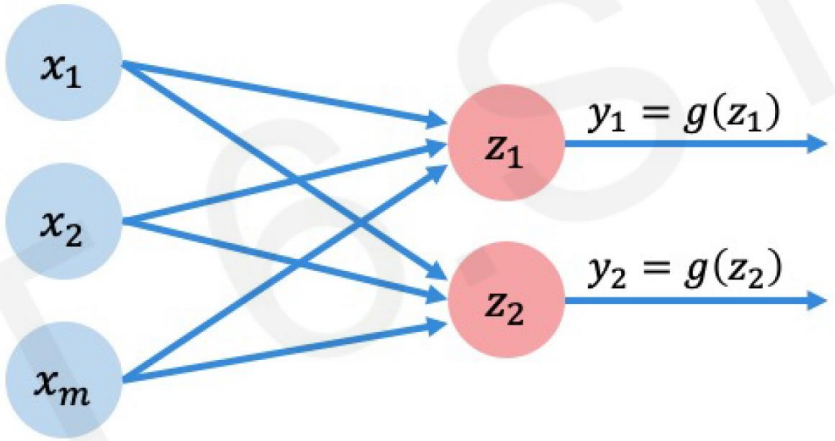


Co-funded by the European Union

Multi Output Perceptron

Multiple perceptrons combine to form layers of a neural network

Because all inputs are densely connected to all outputs, these layers are called **Dense** layers

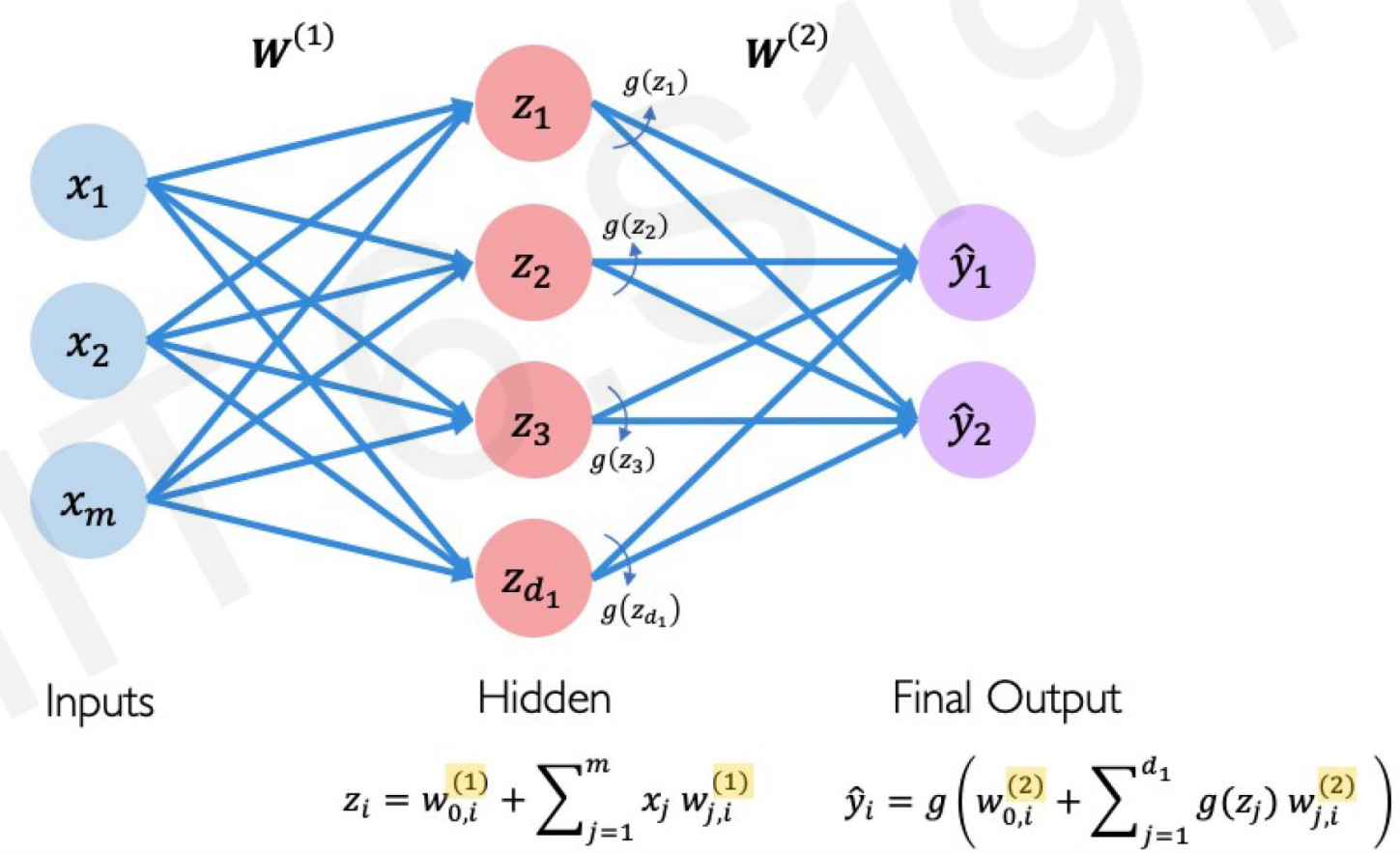


$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

Co-funded by the European Union

Single Layer Neural Network

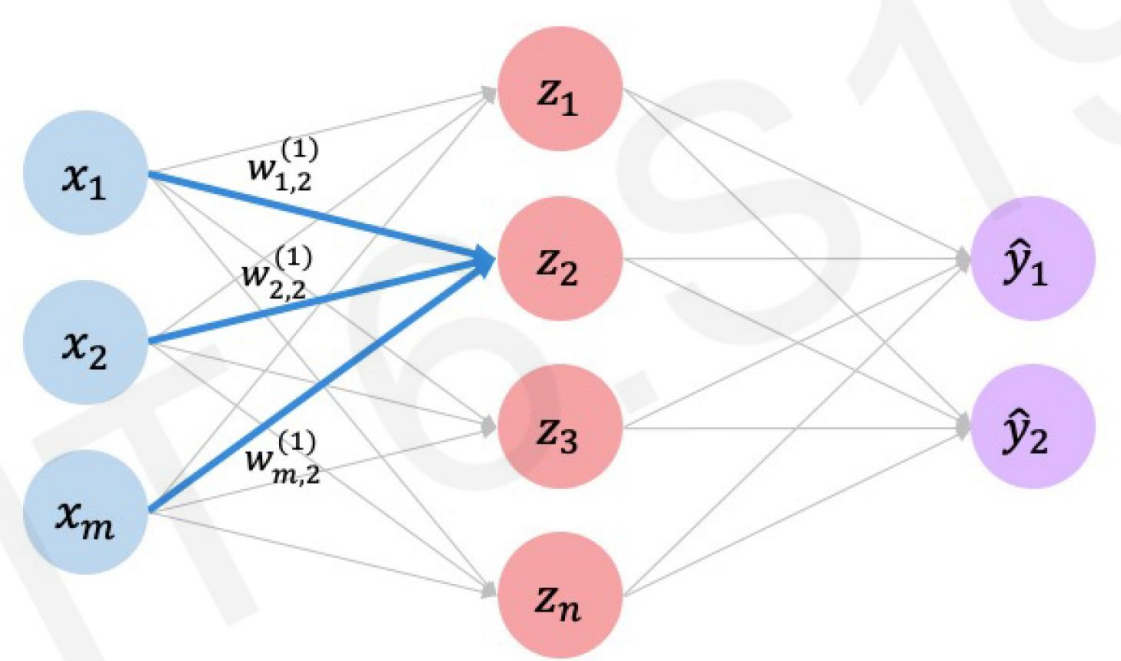
Each connection between neurons has its own weight that gets learned during training



Co-funded by the European Union

Single Layer Neural Network

Each connection between neurons has its own weight that gets learned during training



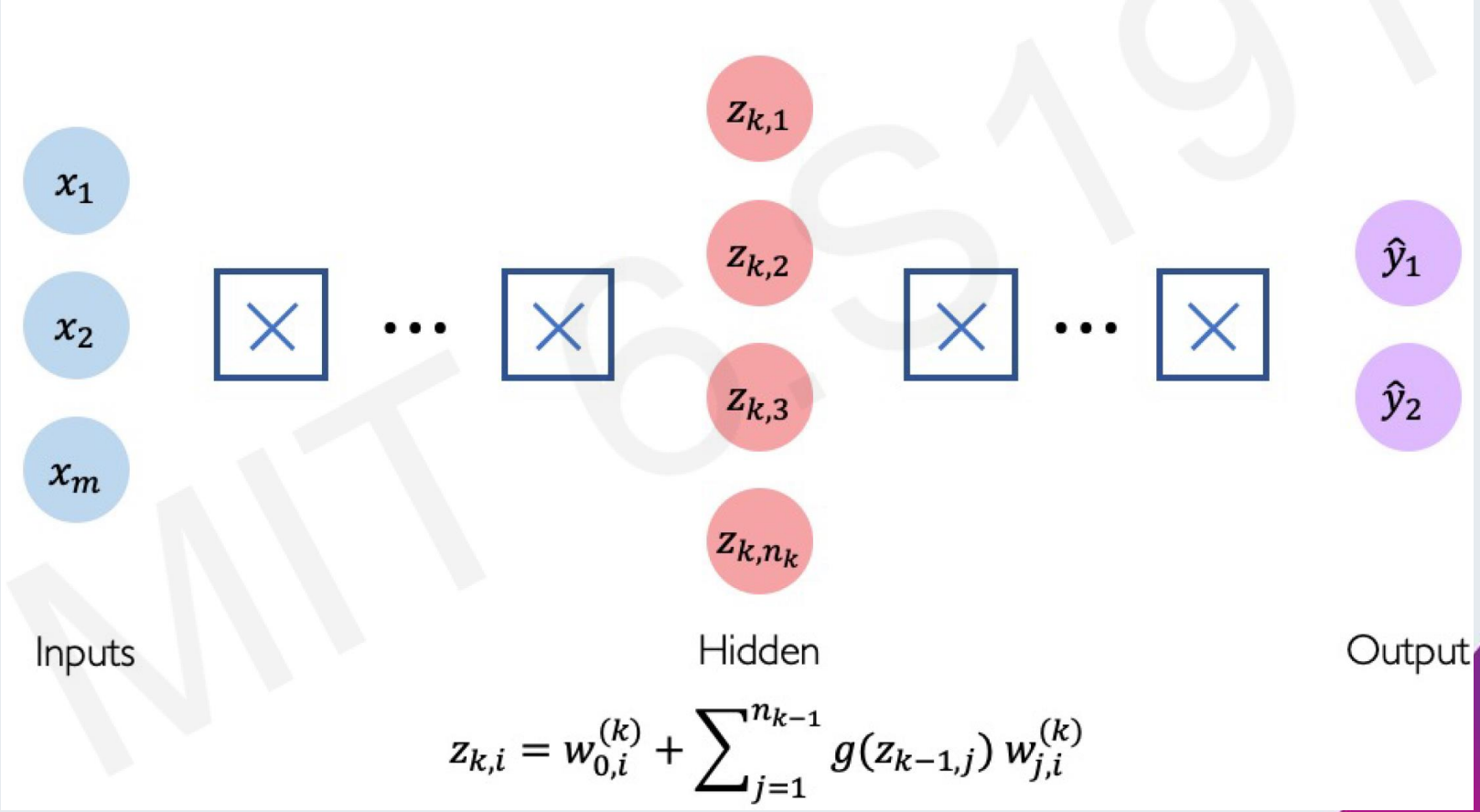
$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,2}^{(1)}$$

$$= w_{0,2}^{(1)} + x_1 \, w_{1,2}^{(1)} + x_2 \, w_{2,2}^{(1)} + x_m \, w_{m,2}^{(1)}$$

Co-funded by the European Union

Deep Neural Network

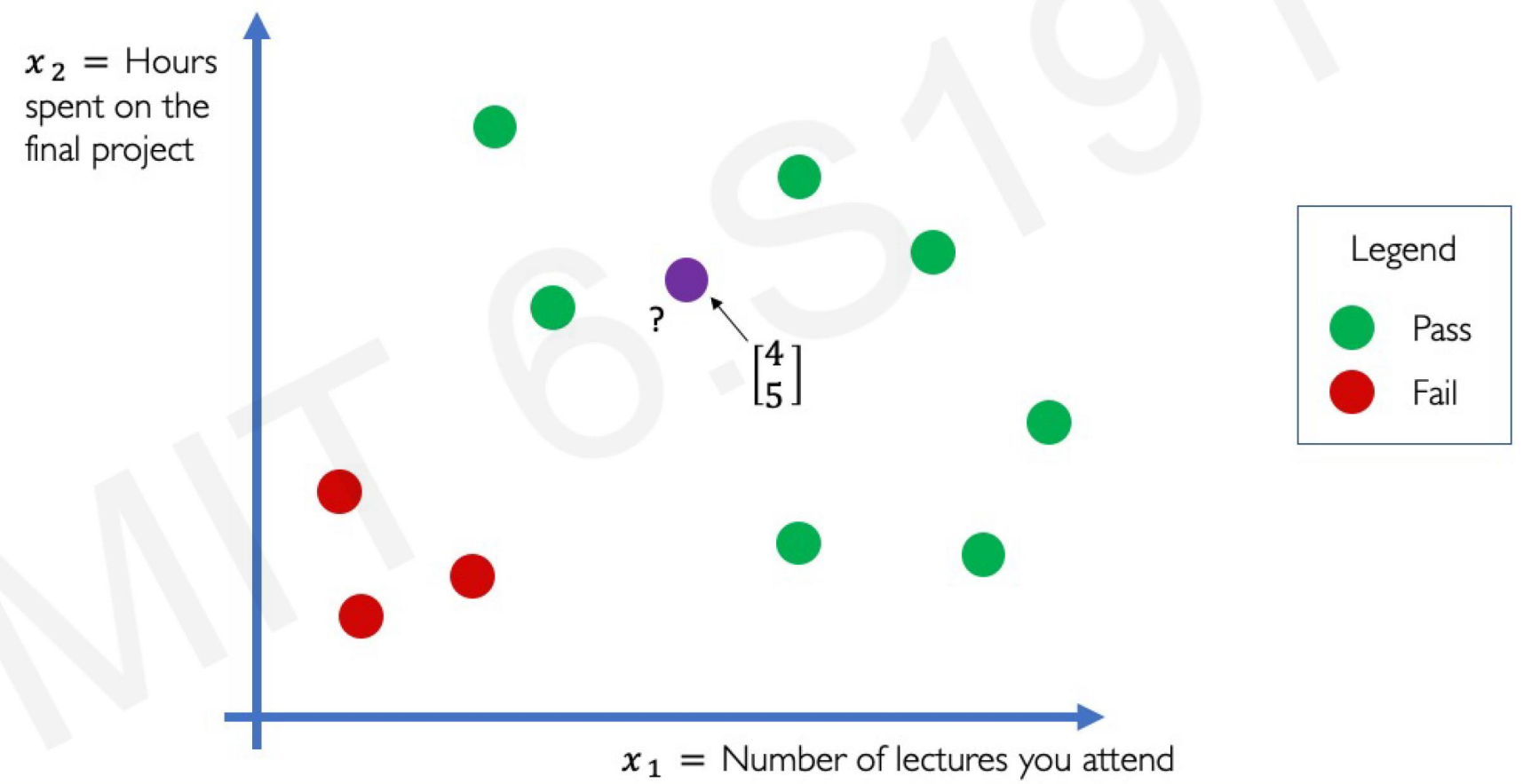
- A single-layer neural network can model only simple linear patterns, whereas a deep neural network can capture highly complex patterns.
- Each connection between neurons has a weight, which is learned through training using optimization techniques like gradient descent.



Co-funded by the European Union

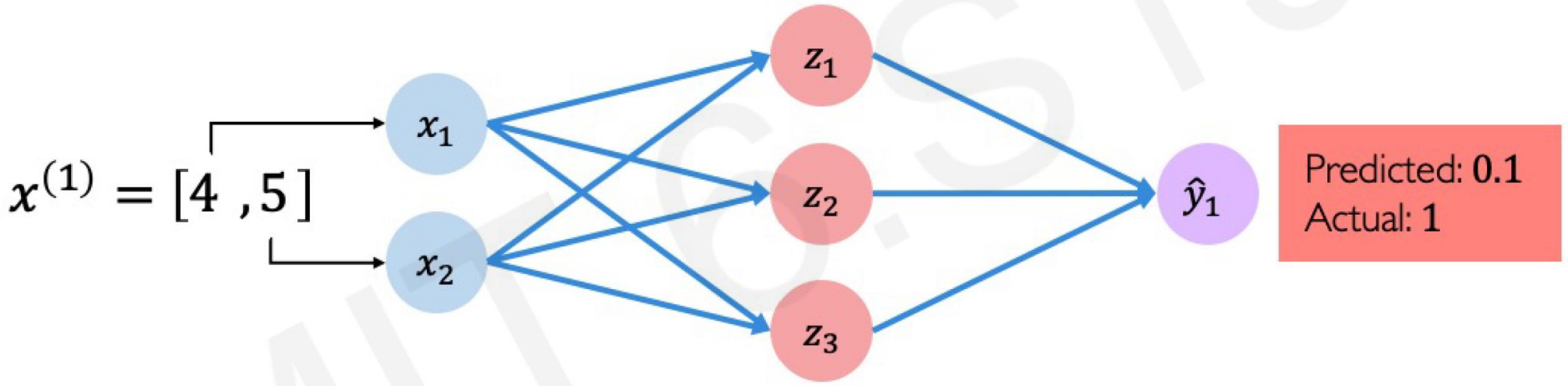
Example Problem: Will I Pass This Class?

The network's depth (number of layers) determines how complex patterns it can learn



Example Problem: Will I Pass This Class?

*The **loss** of our network measures the cost incurred from incorrect predictions*



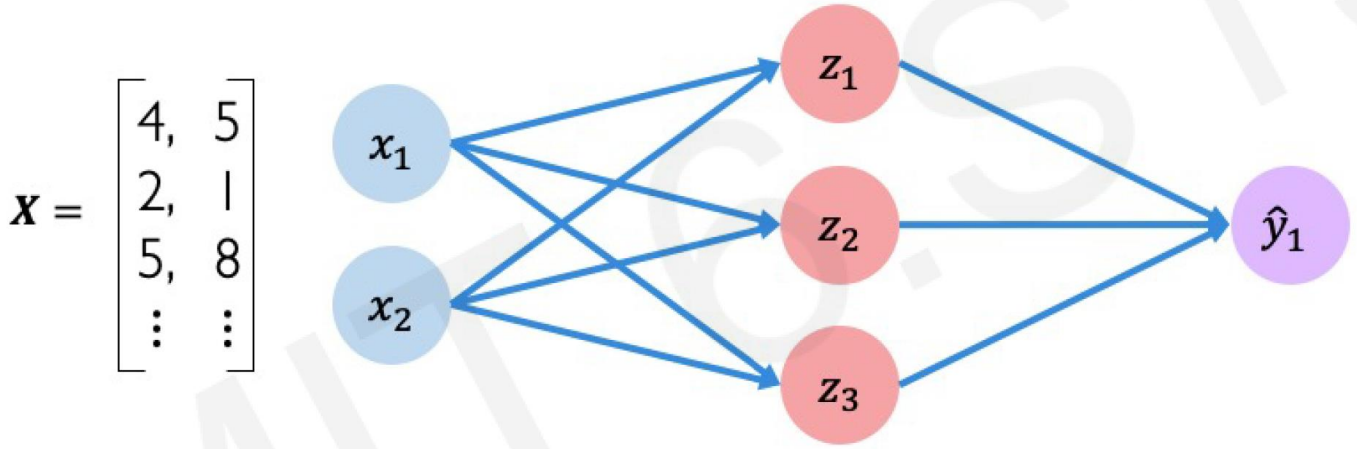
$$\mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

Predicted Actual

Co-funded by the European Union

Empirical Loss

*The **empirical loss** measures the total loss over our entire dataset*



$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$

$f(x)$ y

$$\begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} \quad \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

Also known as:
- Objective function
- Cost function
- Empirical Risk

$$\boldsymbol{J(W)} = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

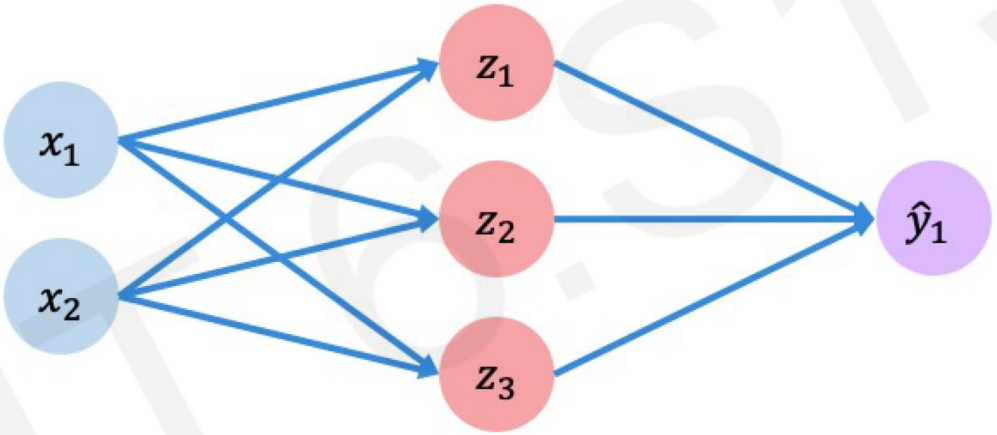
Predicted Actual

Co-funded by the European Union

Binary Cross Entropy Loss

Cross entropy loss *can be used with models that output a probability between 0 and 1*

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$f(x)$ $\quad$ y

$$\begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

$$J(\boldsymbol{W}) = -\frac{1}{n}\sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actual}} \log \underbrace{\left(f\left(x^{(i)}; \boldsymbol{W}\right)\right)}_{\text{Predicted}} + (1 - \underbrace{y^{(i)}}_{\text{Actual}}) \log \underbrace{\left(1 - f\left(x^{(i)}; \boldsymbol{W}\right)\right)}_{\text{Predicted}}$$

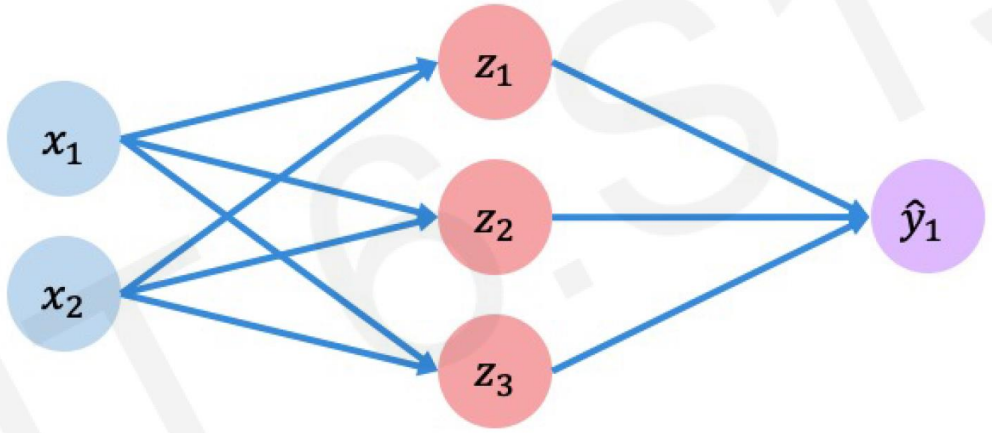
```
loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(y, predicted) )
```

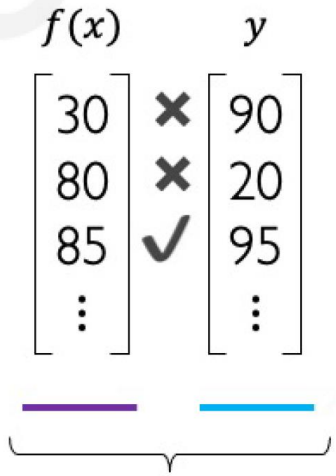
Co-funded by the European Union

Mean Squared Error Loss

Mean squared error loss *can be used with regression models that output continuous real numbers*

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$





$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - f\left(x^{(i)}; \boldsymbol{W}\right)\right)^2$$

Actual Predicted

```
loss = tf.reduce_mean( tf.square(tf.subtract(y, predicted)) )
loss = tf.keras.losses.MSE( y, predicted )
```

Co-funded by the European Union

Digital 4 Business
Evolving your digital future

Training Neural Networks

Co-funded by the European Union

Loss Optimization

We want to find the network weights that ***achieve the lowest loss***

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} J(\boldsymbol{W})$$

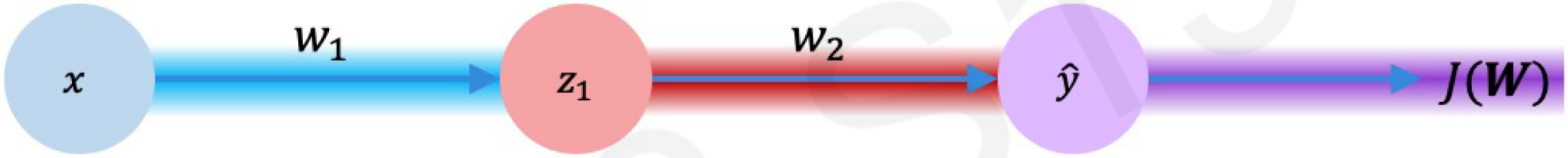
Remember:

$$\boldsymbol{W} = \left\{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}, \dots\right\}$$

Co-funded by the European Union

Computing Gradients: Backpropagation

Backpropagation calculates how much each weight contributed to the error



$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Repeat this for ***every weight in the network*** *using gradients from later layers*

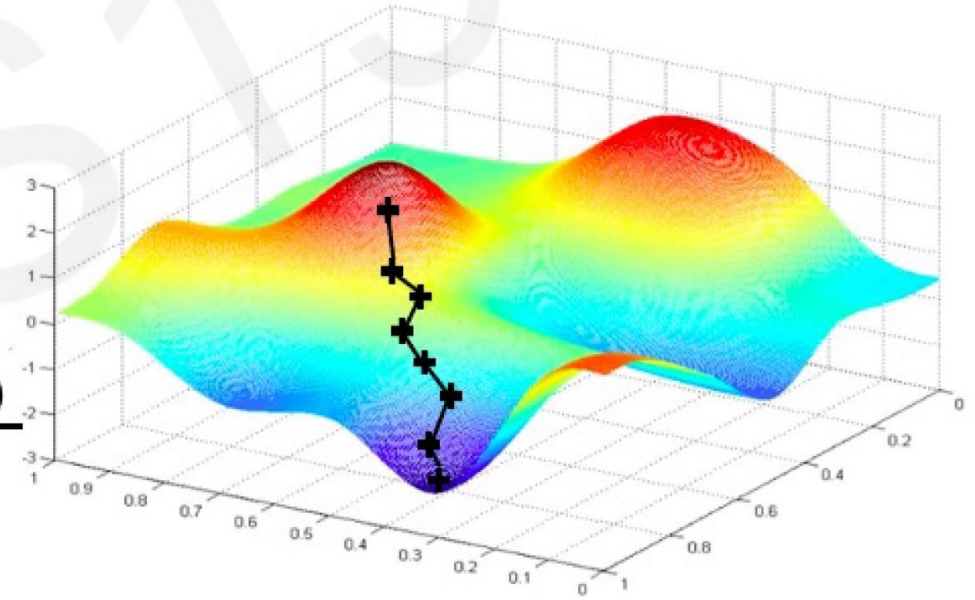
Co-funded by the European Union

Stochastic Gradient Descent

Gradient descent updates weights in the direction that reduces error

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(W)}{\partial W} = \frac{1}{B} \sum_{k=1}^{B} \frac{\partial J_k(W)}{\partial W}$
5. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
6. Return weights



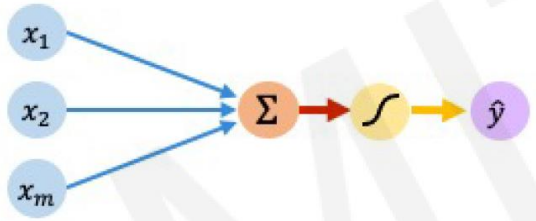
Co-funded by the European Union

Neural Networks Summary

Training requires finding the right balance between learning rate, batch size, and number of epochs

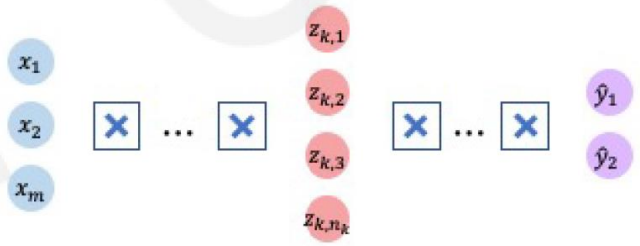
The Perceptron

- Structural building blocks
- Nonlinear activation functions



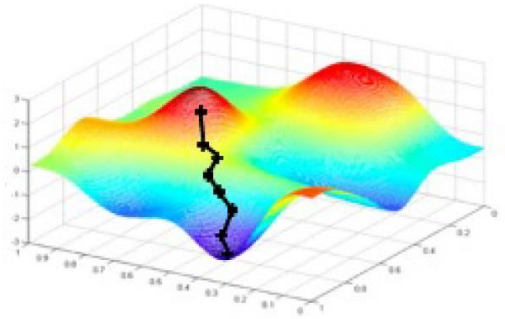
Neural Networks

- Stacking Perceptrons to form neural networks
- Optimization through backpropagation



Training in Practice

- Adaptive learning
- Batching
- Regularization



Co-funded by the European Union

Deep Learning for Computer Vision

Digital 4 Business
Evolving your digital future

Co-funded by the European Union

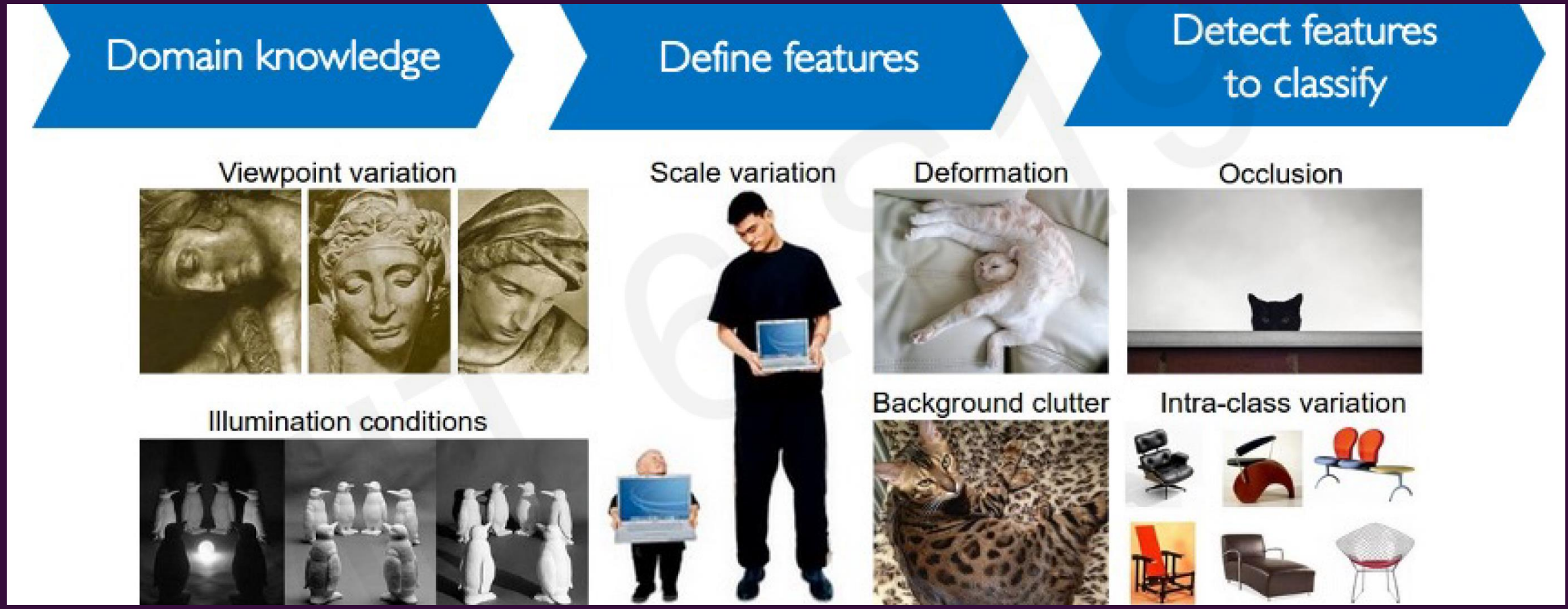
Tasks in Computer Vision

Computer vision tasks require understanding spatial relationships in image data

- **Regression**: output variable takes continuous value
- **Classification**: output variable takes class label. Can produce probability of belonging to a particular class

Co-funded by the European Union

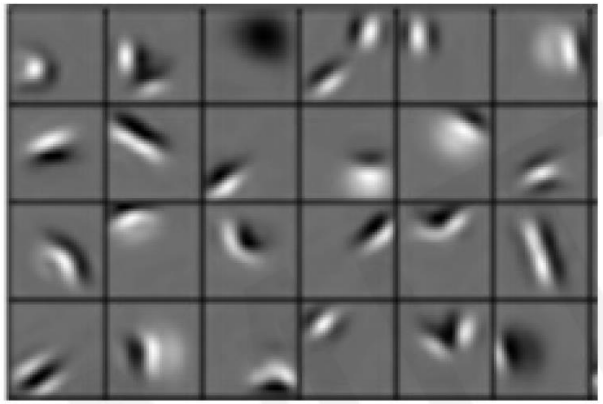
Manual Feature Extraction



Learning Feature Representations

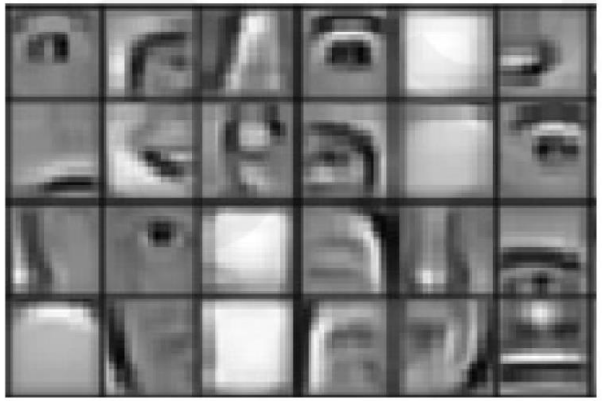
Can we learn a **hierarchy of features** directly from the data instead of hand engineering?

Low level features



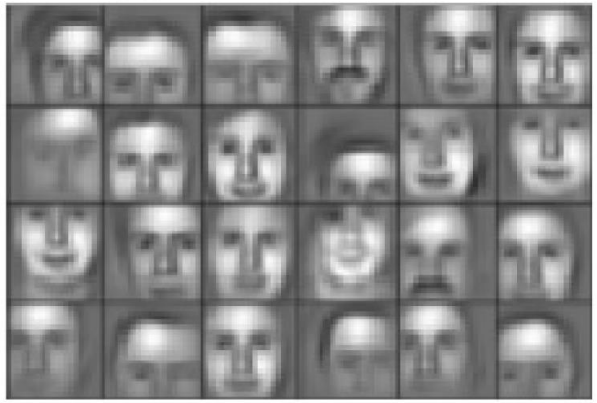
Edges, dark spots

Mid level features



Eyes, ears, nose

High level features



Facial structure

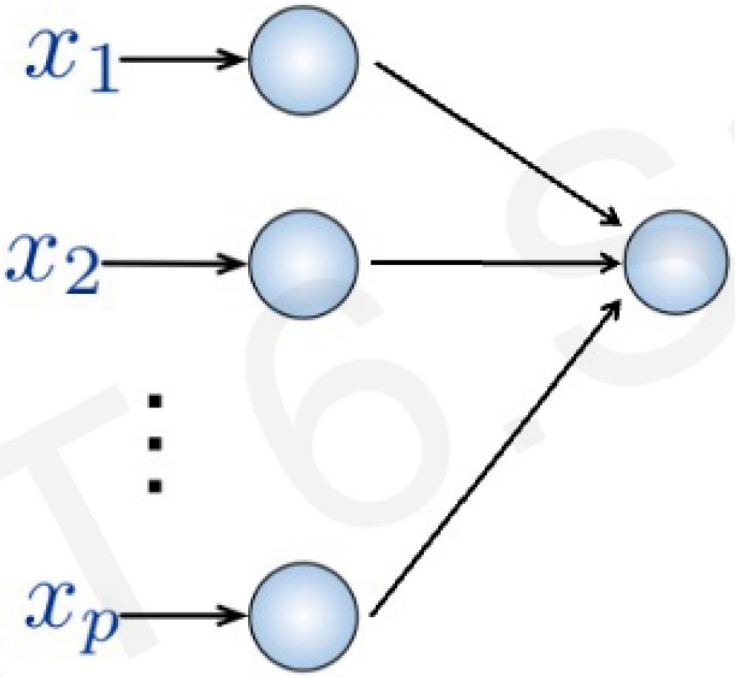
Co-funded by the European Union

Fully Connected Neural Network

Traditional neural networks don't handle spatial data efficiently

Input:

- 2D image
- Vector of pixel values



Fully Connected:

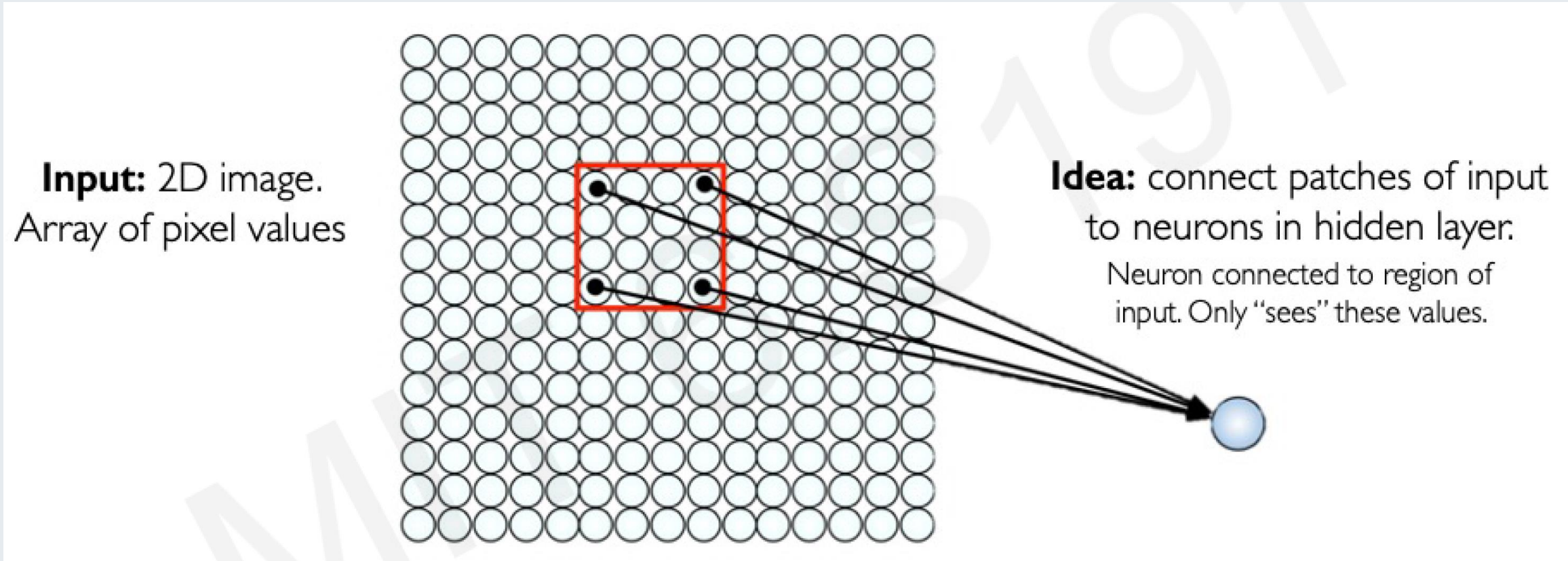
- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

How can we use **spatial structure** in the input to inform the architecture of the network?

Co-funded by the European Union

Using Spatial Structure

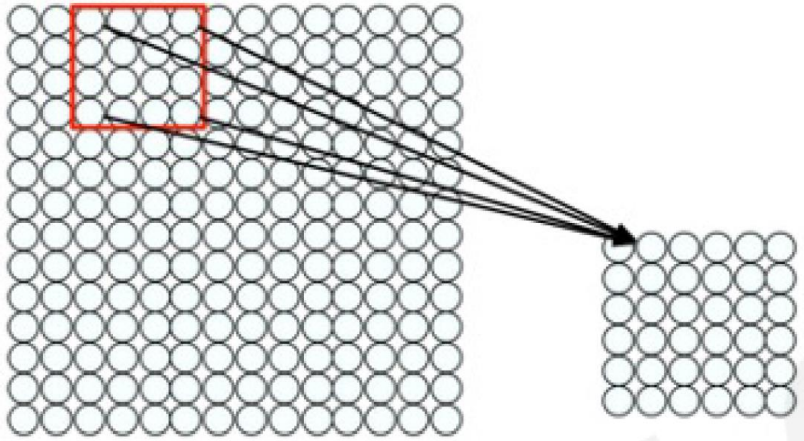
Convolutional layers learn filters that detect features like edges, textures, and patterns



Co-funded by the European Union

Feature Extraction with Convolution

The network progressively learns more complex visual features in deeper layers



- Filter of size 4x4 : 16 different weights
- Apply this same filter to 4x4 patches in input
- Shift by 2 pixels for next patch

This "patchy" operation is **convolution**

1) Apply a set of weights – a filter – to extract **local features**

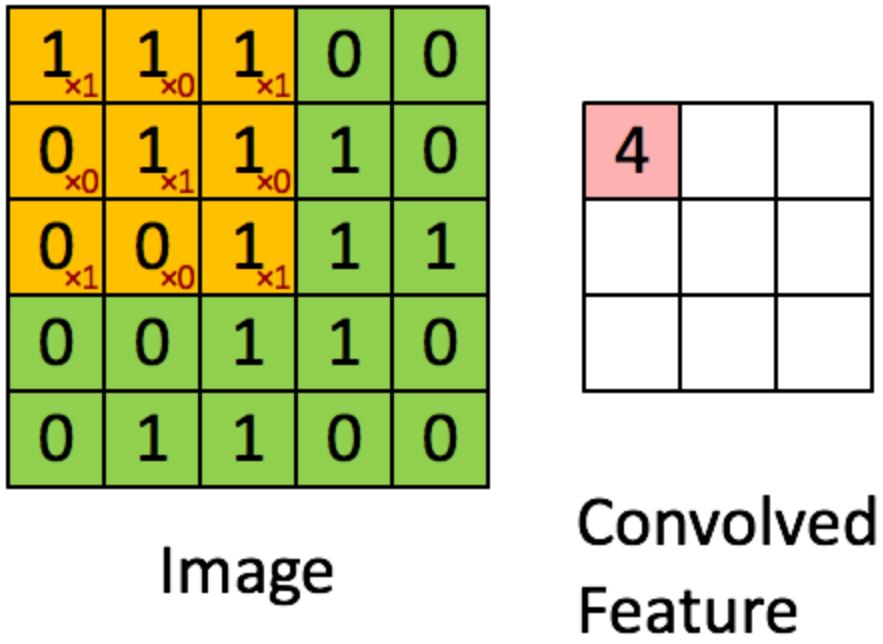
2) Use **multiple filters** to extract different features

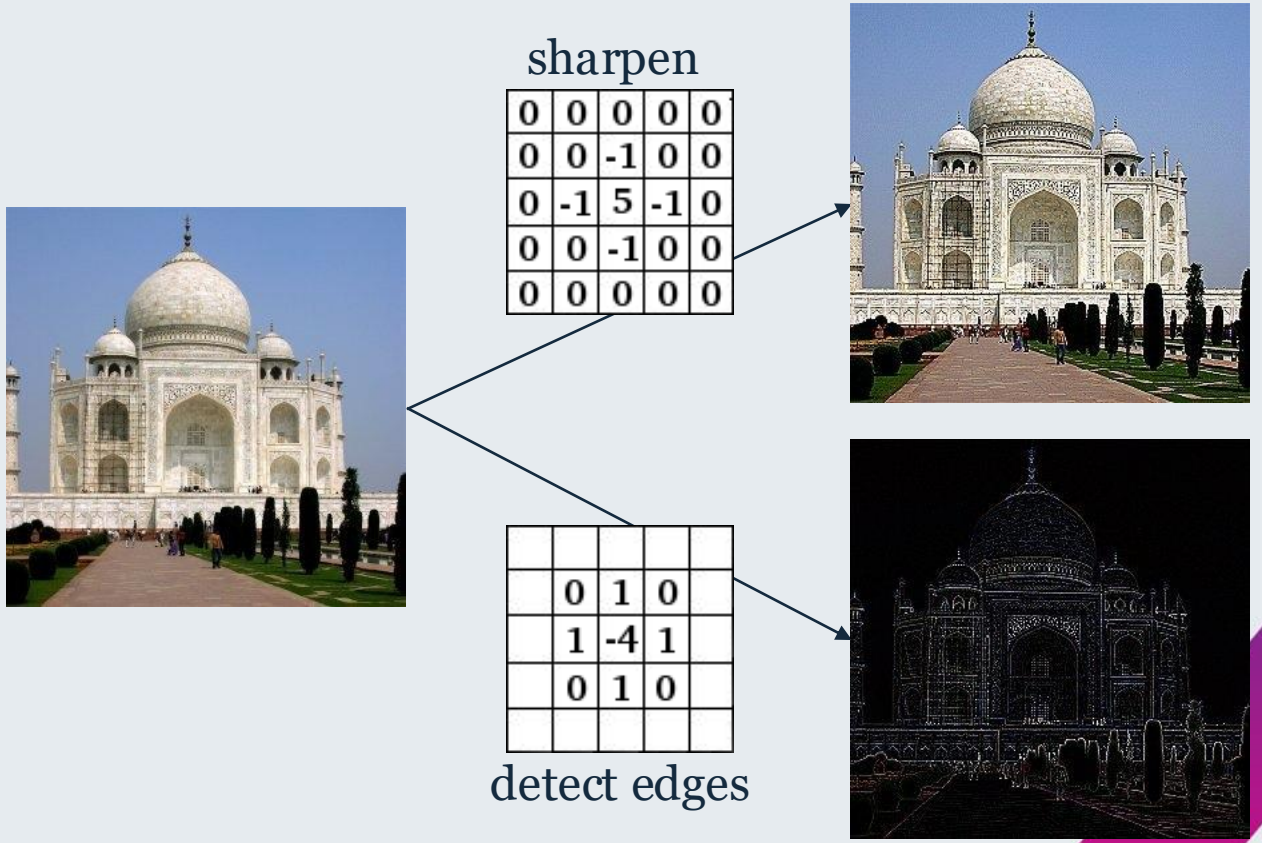
3) **Spatially share** parameters of each filter

Co-funded by the European Union

Convolutional Neural Networks

Key components include filters (kernels), pooling layers, and fully connected layers for classification.

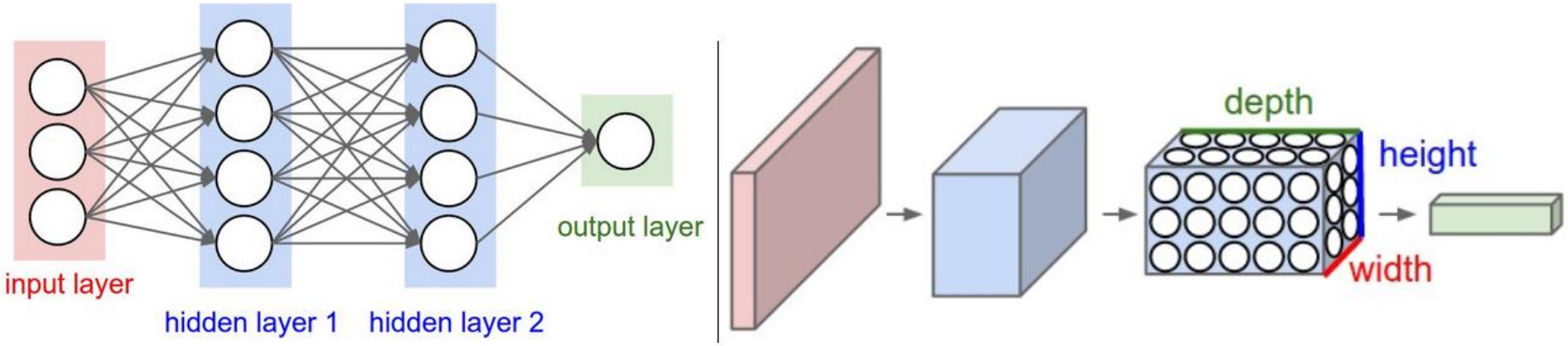




http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

Convolutional Neural Networks

- http://cs231n.github.io/convolutional-networks/

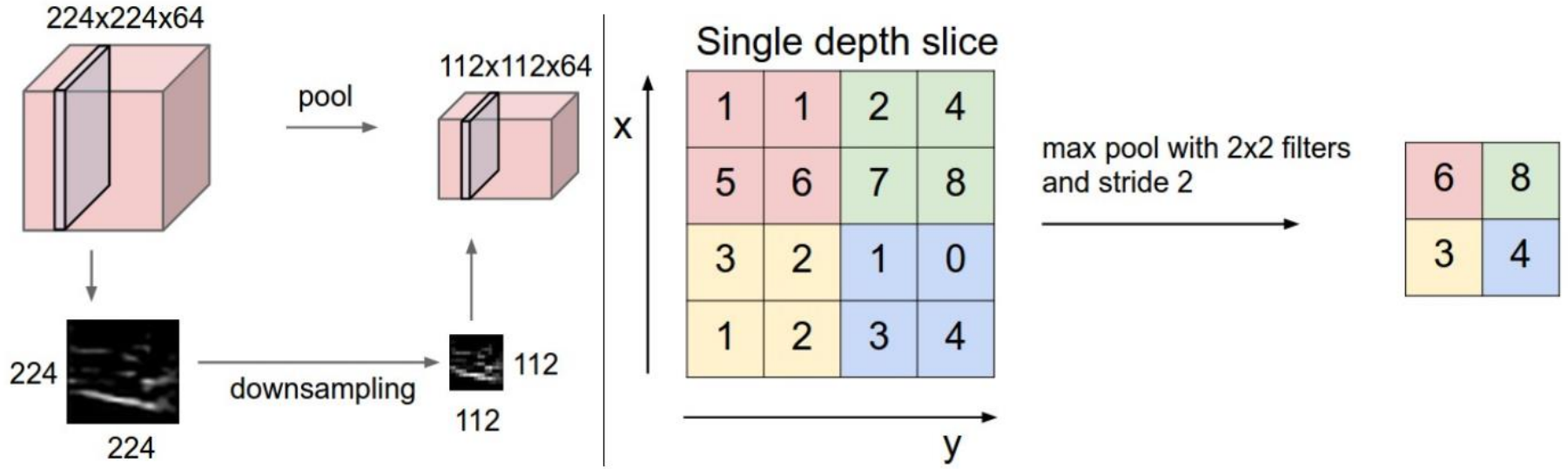


Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Co-funded by the European Union

Convolutional Neural Networks

- http://cs231n.github.io/convolutional-networks/

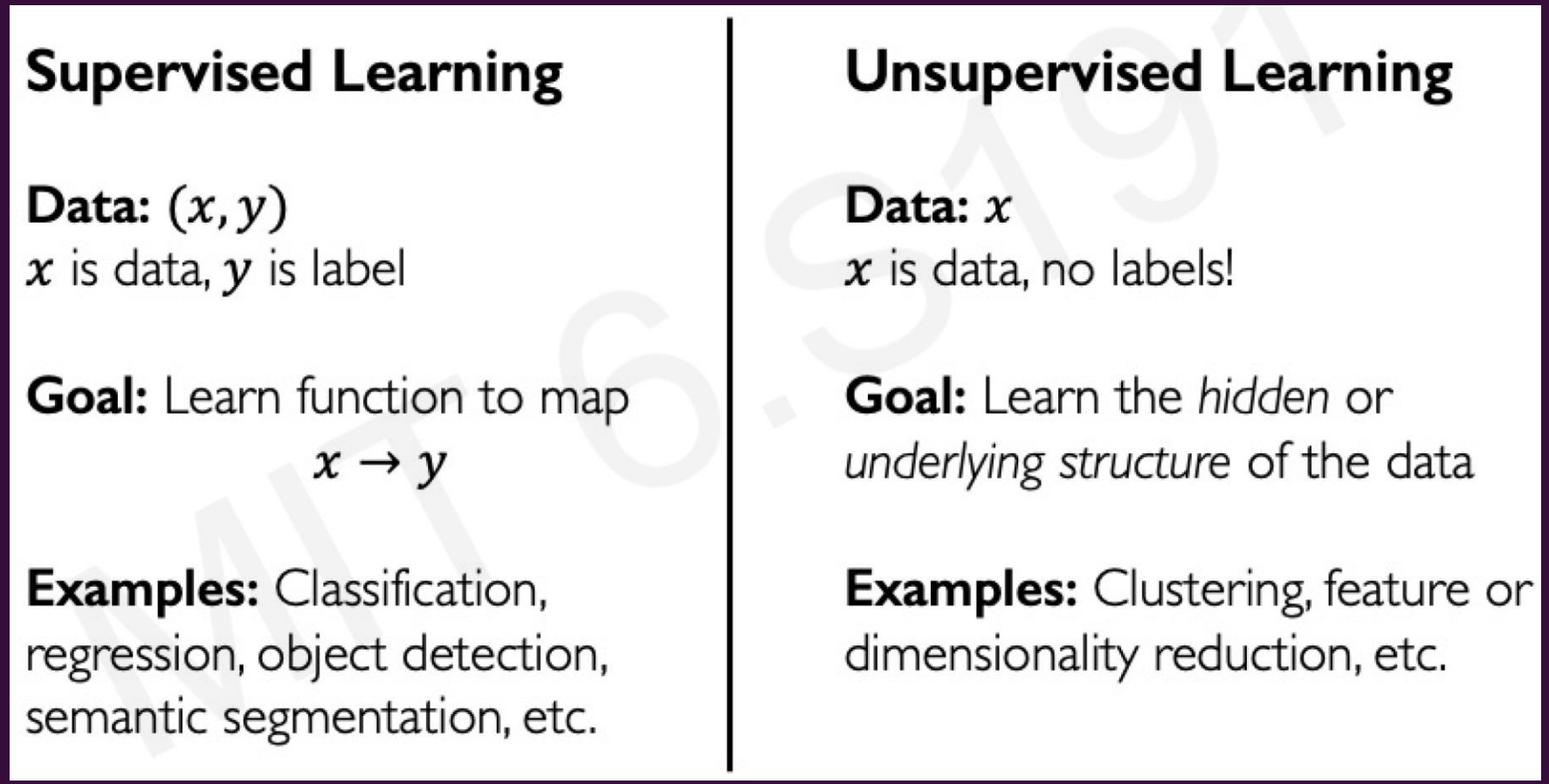


Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. **Left:** In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. **Right:** The most common downsampling operation is max, giving rise to **max pooling**, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).

Co-funded by the European Union

Supervised vs Unsupervised

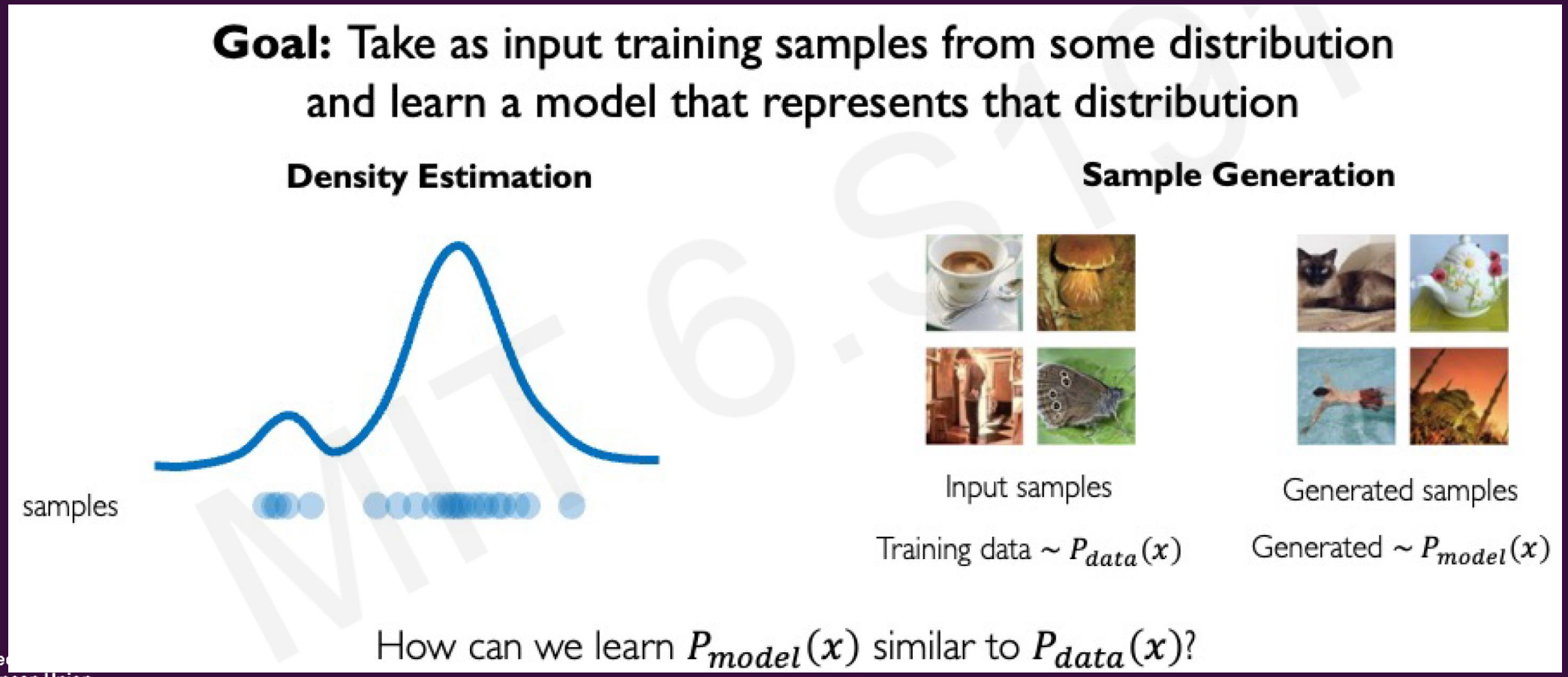
- Supervised learning uses labeled data for training, while unsupervised learning identifies patterns in unlabeled data.
- Deep generative models, such as Autoencoders and Generative Adversarial Networks (GANs), are commonly used for unsupervised learning.



Co-funded by the European Union

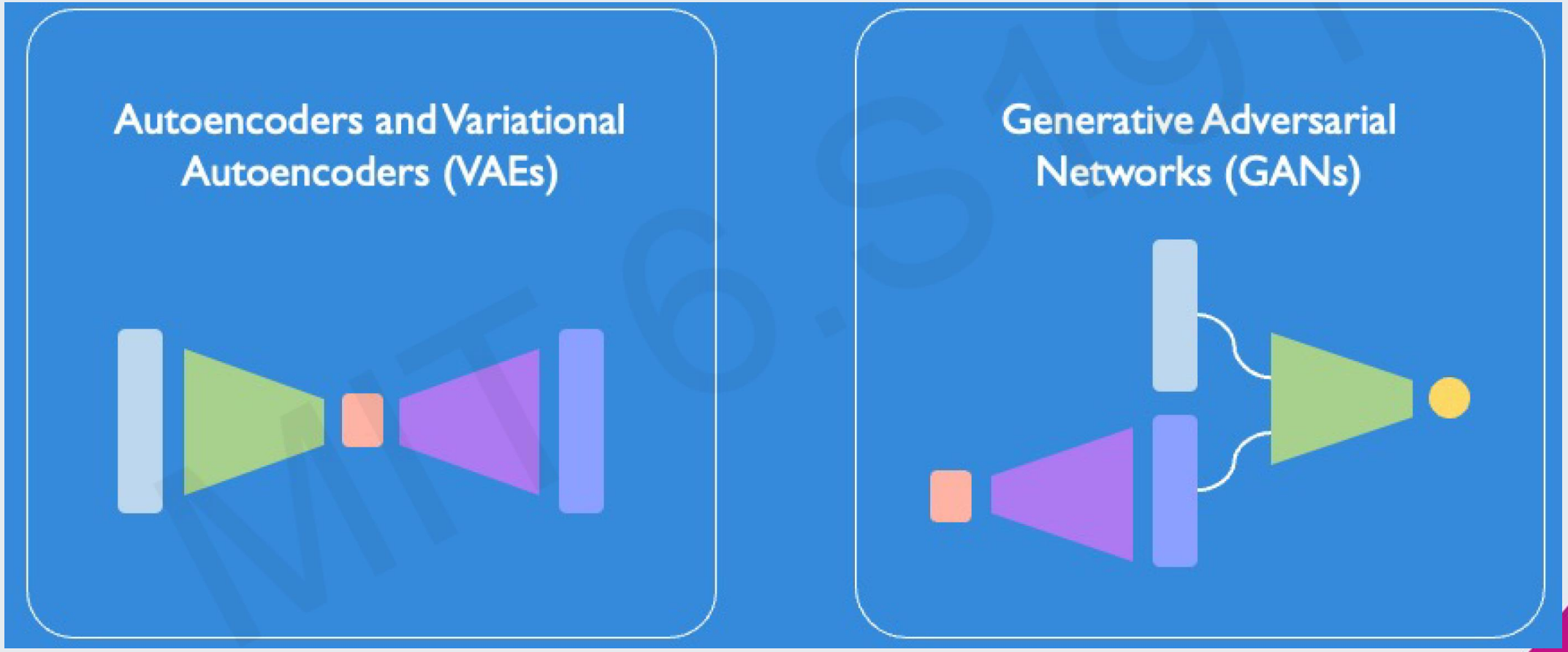
Generative Modeling

Generative models learn the underlying distribution of data to create new samples that resemble the original dataset.



Co-funded by the European Union

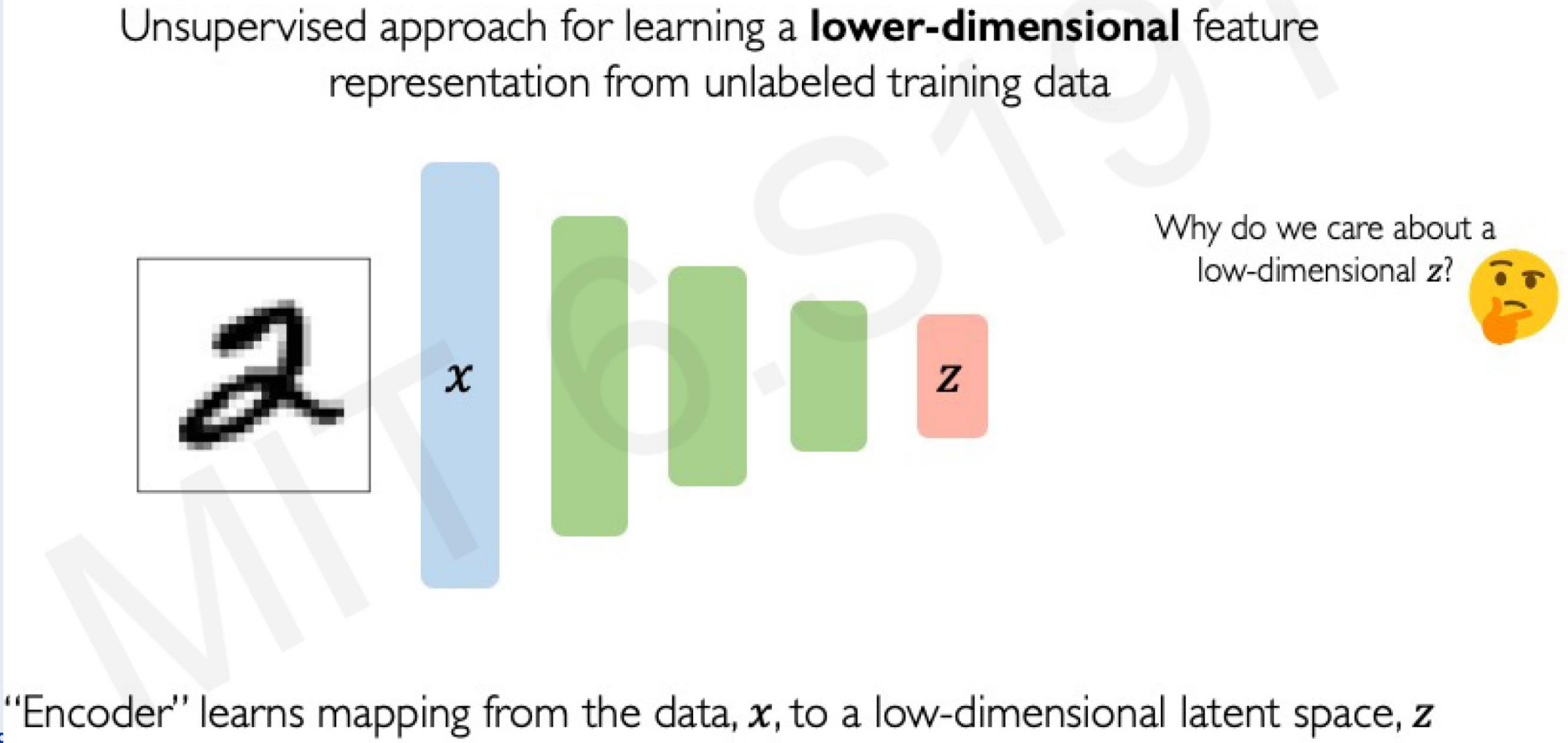
Latent Variable Models



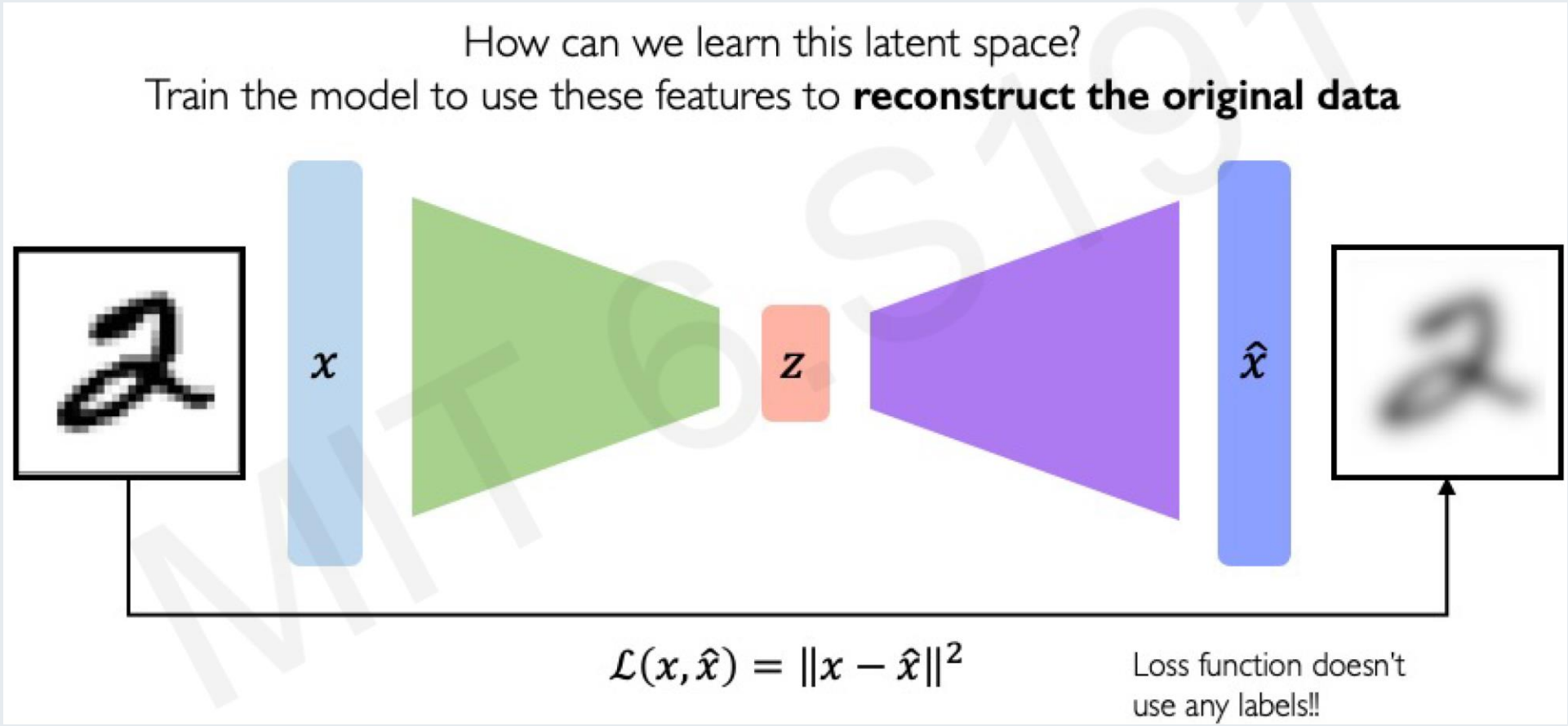
Co-funded by the European Union

Autoencoders: Background

Autoencoders compress input data into a smaller representation (latent space) and reconstruct it, often used for anomaly detection and feature learning.



Autoencoders: Background



Co-funded by the European Union

Dimensionality of Latent Space -> Reconstruction Quality

Autoencoding is a form of compression!
Smaller latent space will force a larger training bottleneck

2D latent space

5D latent space

Ground Truth

Co-funded by
the European Union

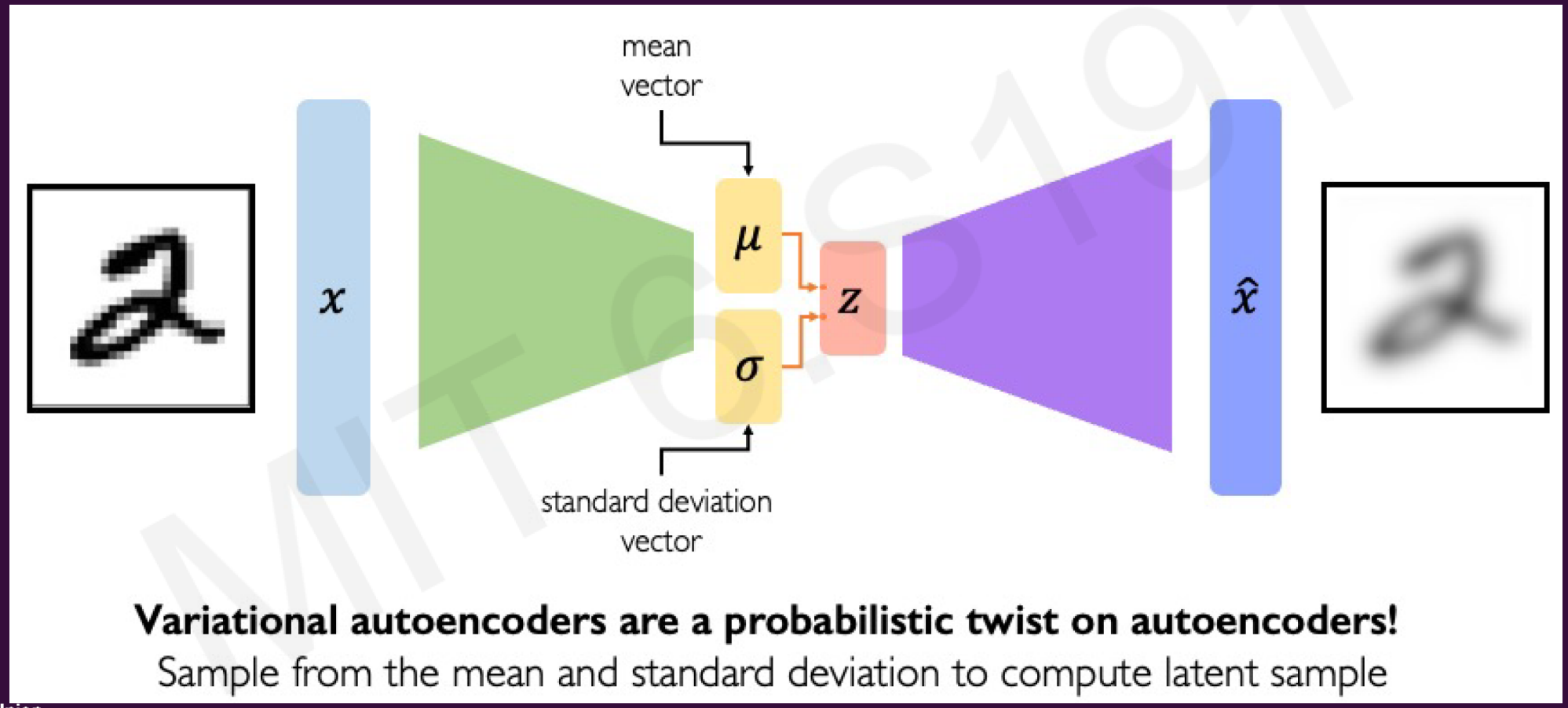
Variational Autoencoder (VAE)

Digital 4 Business
Evolving your digital future

Co-funded by the European Union

VAEs: Key Difference with Traditional Autoencoder

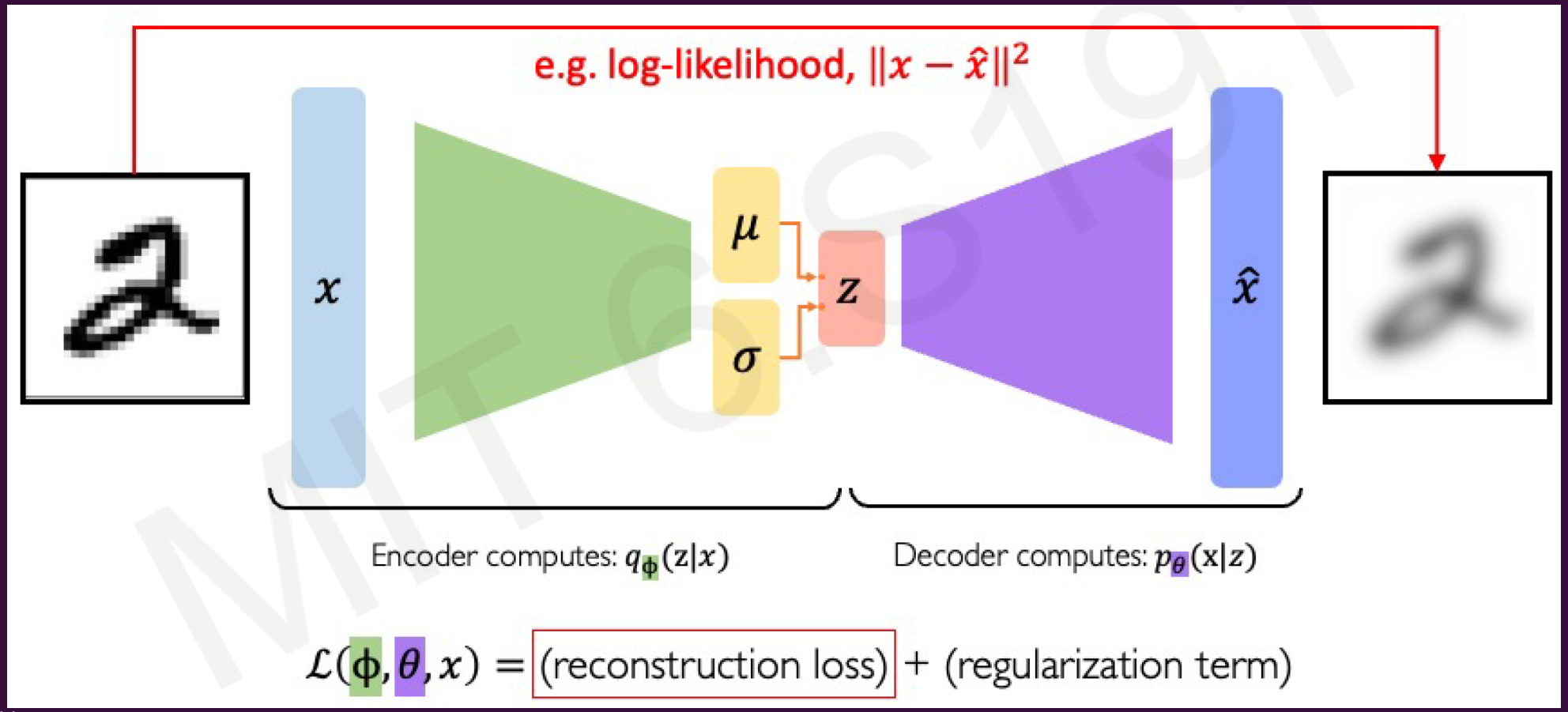
VAEs extend traditional autoencoders by introducing probabilistic modeling, enabling more structured and meaningful latent representations.



Co-funded by
the European Union

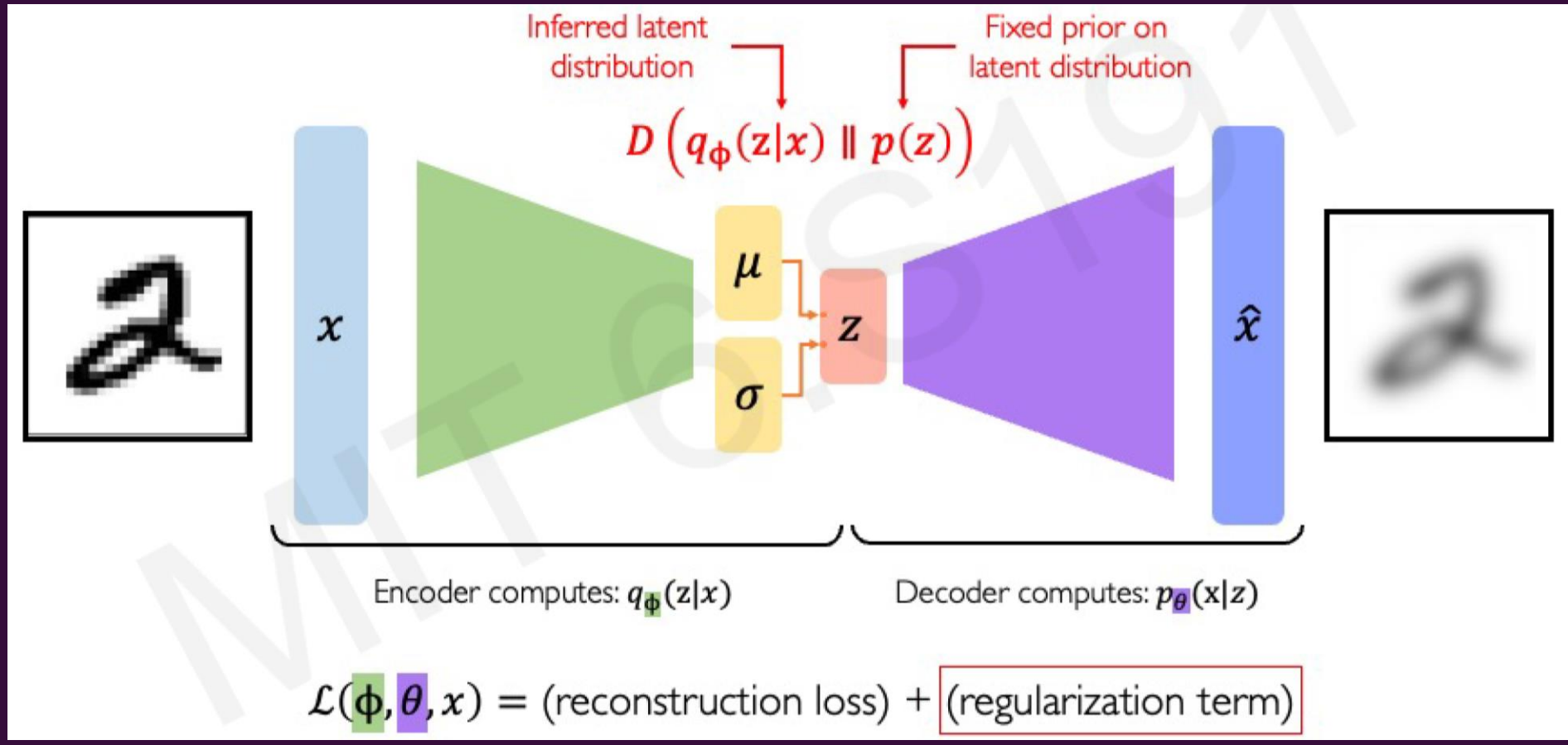
VAE Optimization

They optimize a balance between reconstruction accuracy and latent space regularization, making them useful for data generation.



VAE Optimization

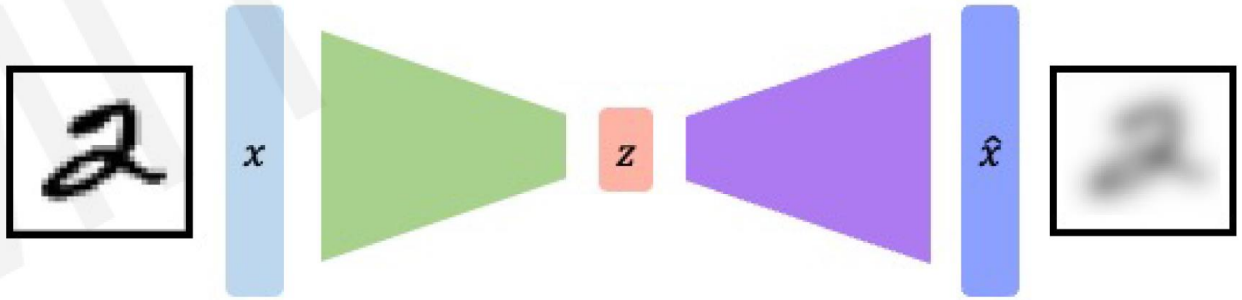
- Variational Autoencoders (VAEs) learn compressed representa
- These models enable applications like image synthesis and data augmentationt of data



Co-funded by the European Union

VAE Summary

1. Compress representation of world to something we can use to learn
2. Reconstruction allows for unsupervised learning (no labels!)
3. Reparameterization trick to train end-to-end
4. Interpret hidden latent variables using perturbation
5. Generating new examples



Co-funded by
the European Union

Generative Adversarial Network (GAN)

Digital 4 Business
Evolving your digital future

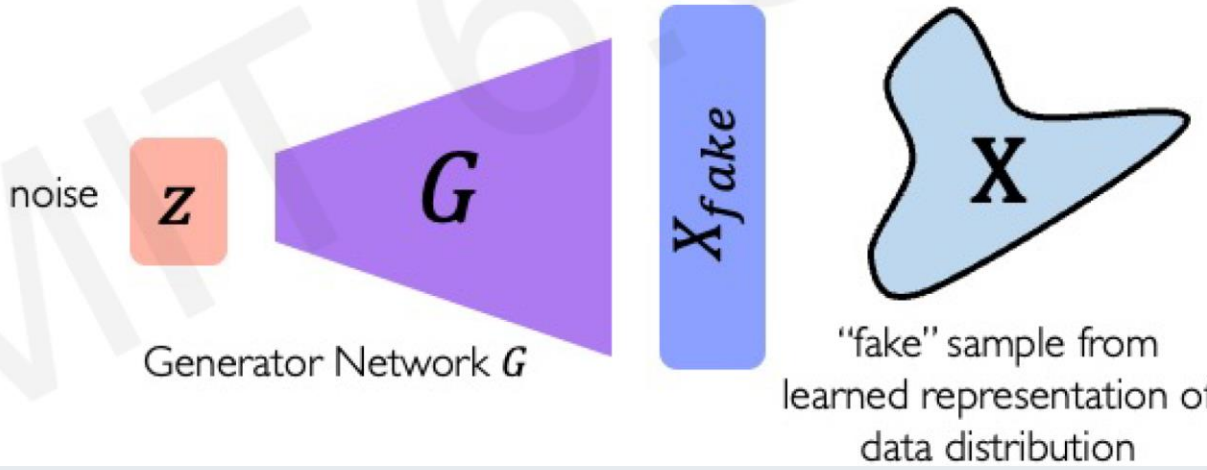
Co-funded by the European Union

What if we Just Want to Sample?

Idea: don't explicitly model density, and instead just sample to generate new instances.

Problem: want to sample from complex distribution – can't do this directly!

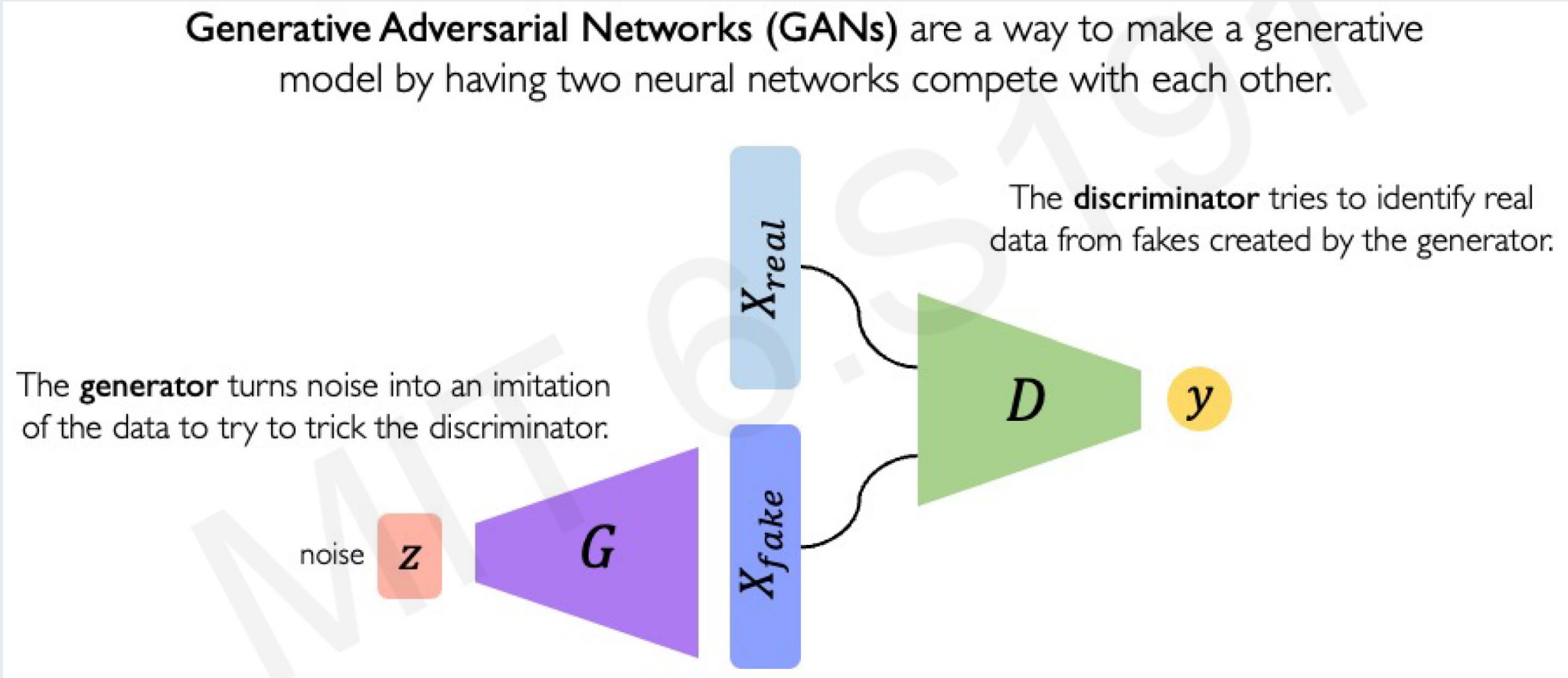
Solution: sample from something simple (e.g., noise), learn a transformation to the data distribution.



Co-funded by the European Union

Generative Adversarial Network (GAN)

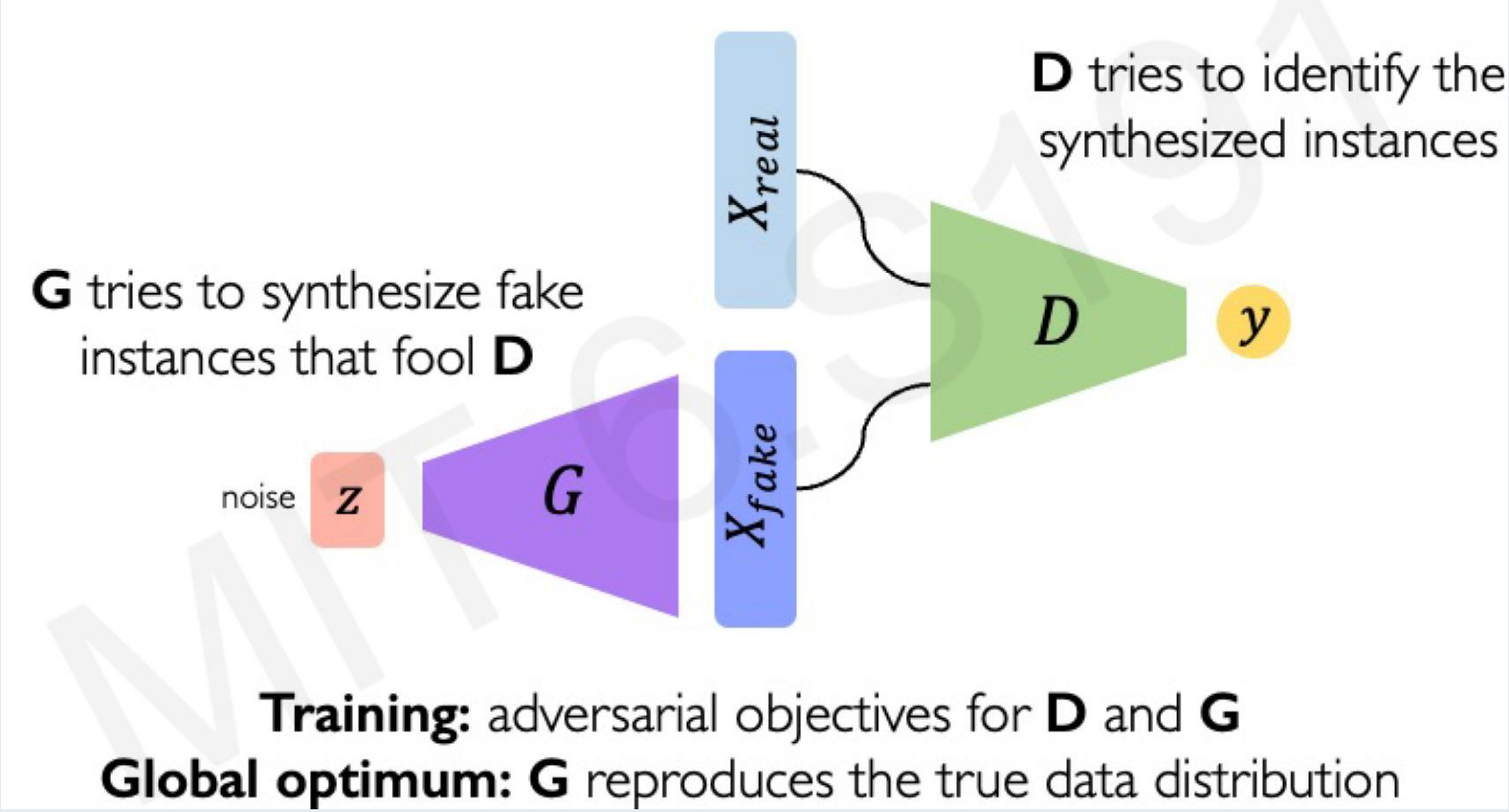
GANs use two competing networks: a generator that creates data and a discriminator that evaluates it



Co-funded by the European Union

Training GANs

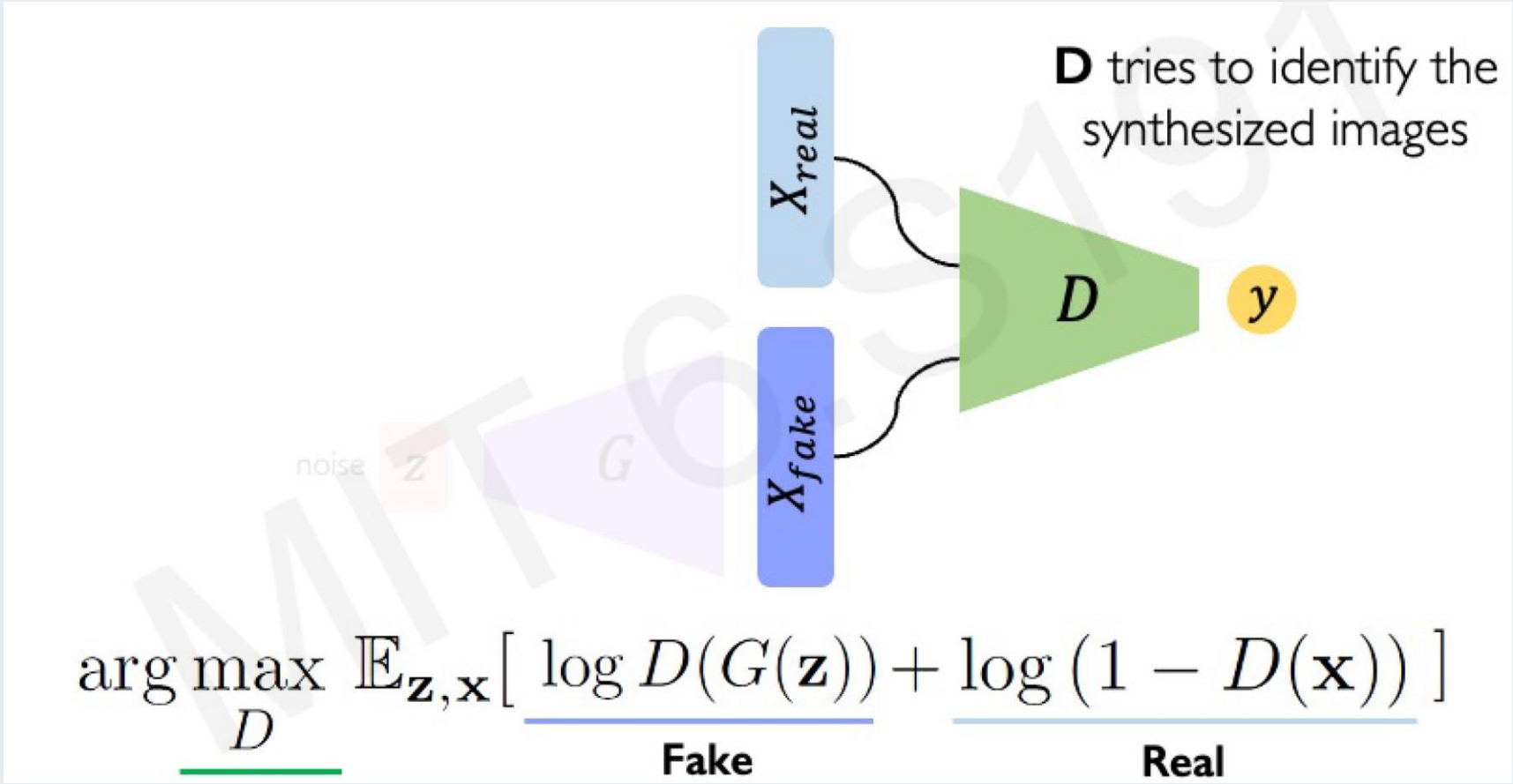
The generator creates fake data, while the discriminator learns to distinguish between real and fake samples, improving both over time.



Co-funded by the European Union

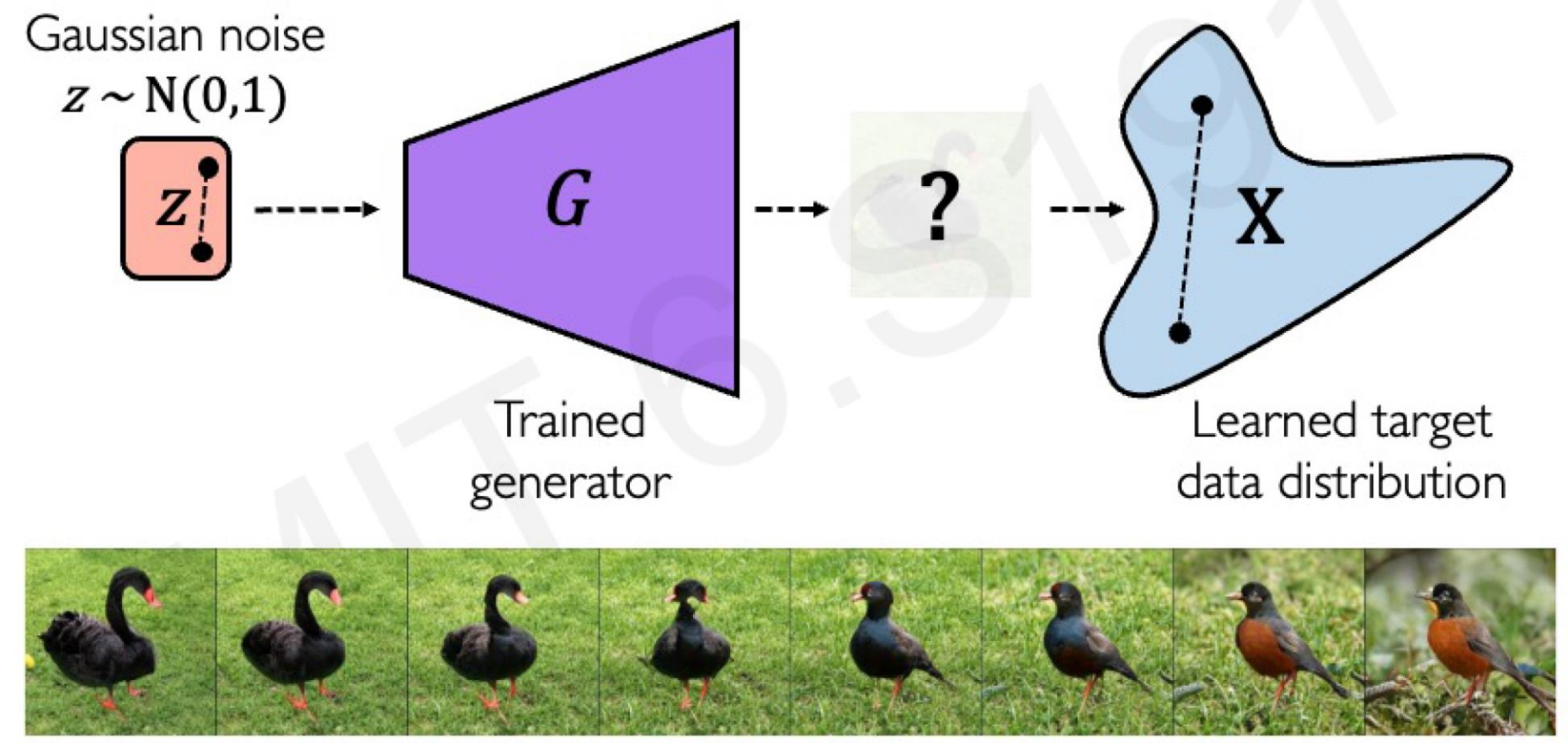
Training GANs: Loss Function

GANs use a min-max game approach where the generator tries to fool the discriminator, and the discriminator aims to correctly classify samples.



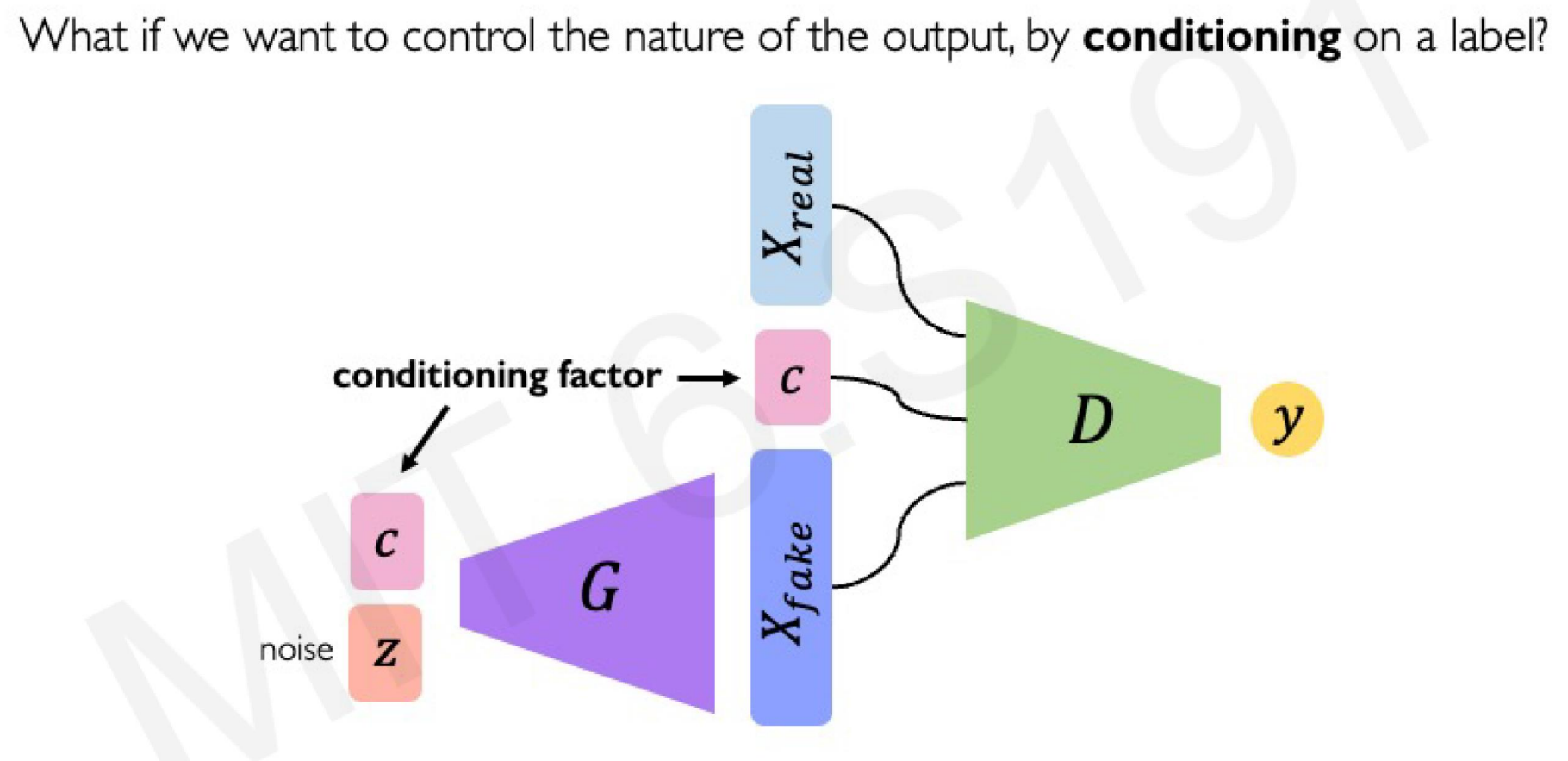
Co-funded by the European Union

GANs are Distribution Transformers



Co-funded by the European Union

Conditional GANs

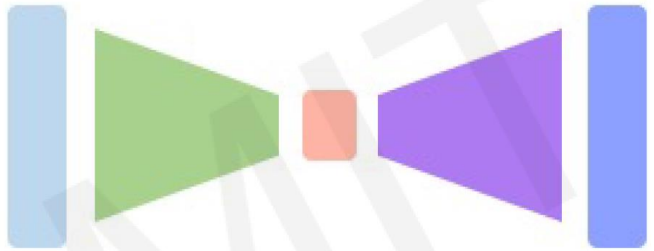


Co-funded by the European Union

Deep Generative Modeling Summary

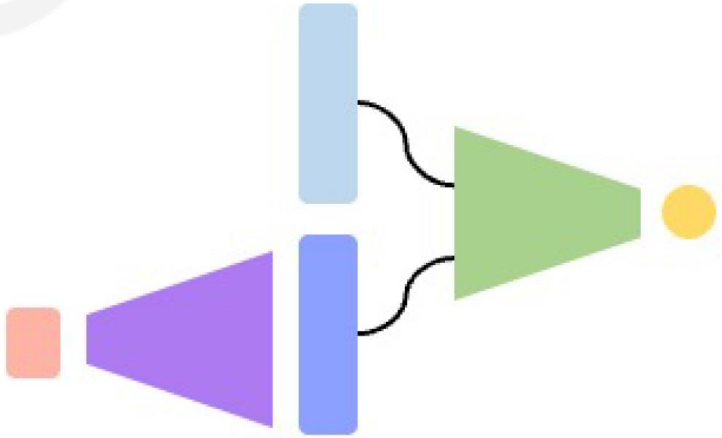
Autoencoders and Variational Autoencoders (VAEs)

Learn lower-dimensional **latent space** and **sample** to generate input reconstructions



Generative Adversarial Networks (GANs)

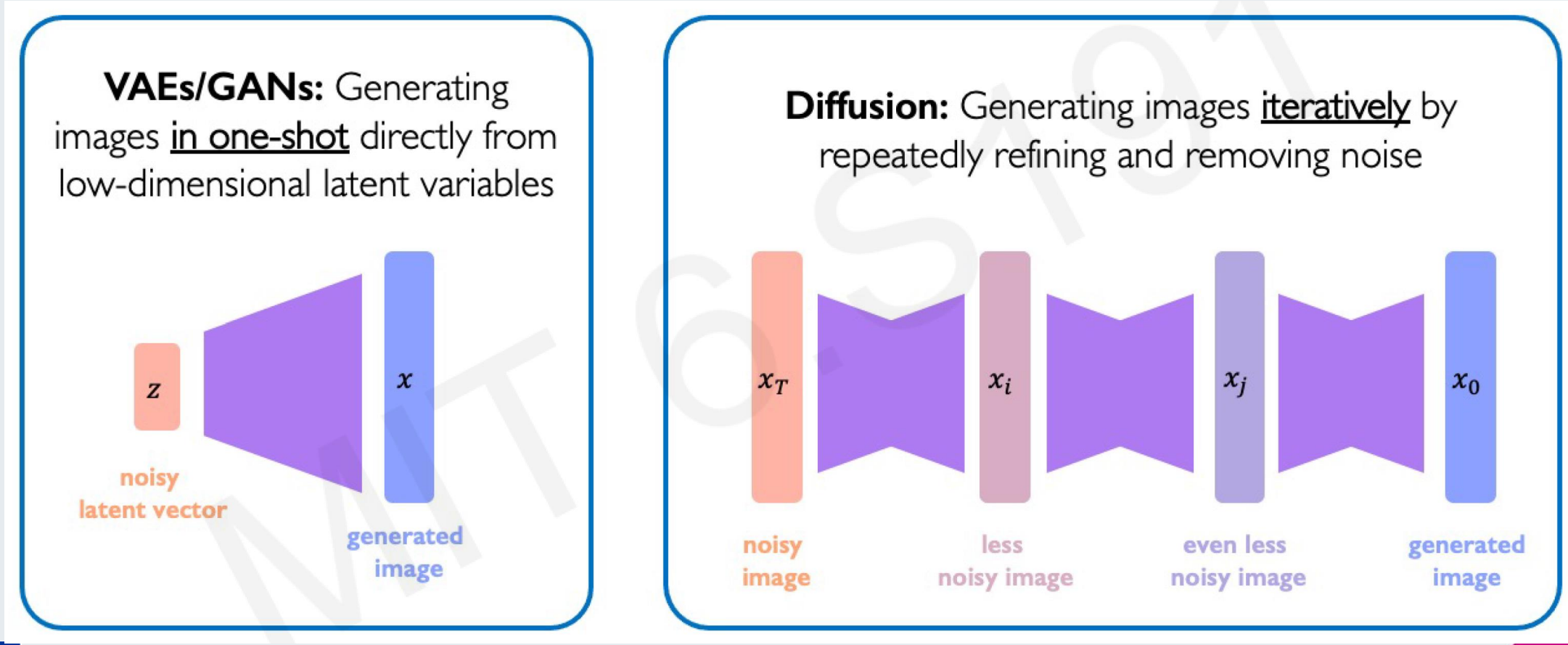
Competing **generator** and **discriminator** networks



Co-funded by the European Union

Diffusion Models

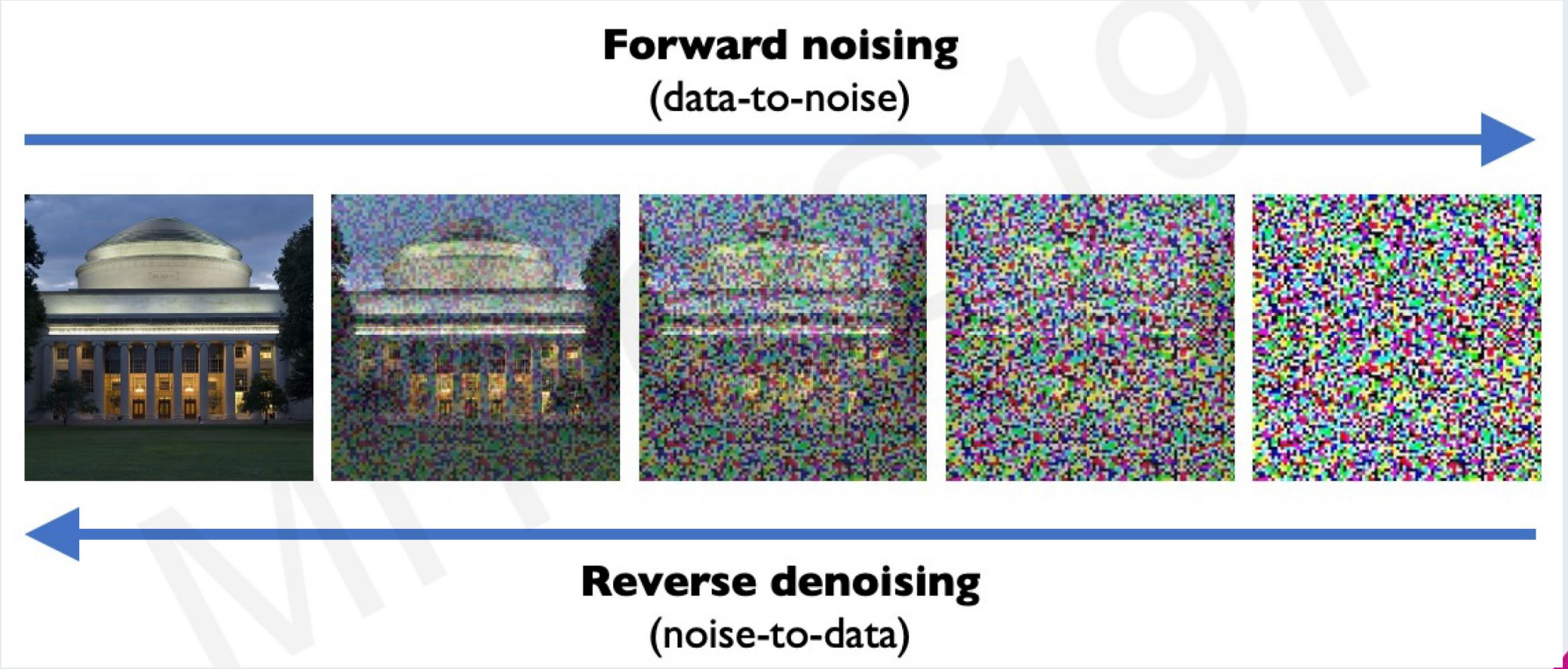
Diffusion models iteratively transform noise into structured data, often used for image synthesis and denoising tasks.



Co-funded by the European Union

The Diffusion Process

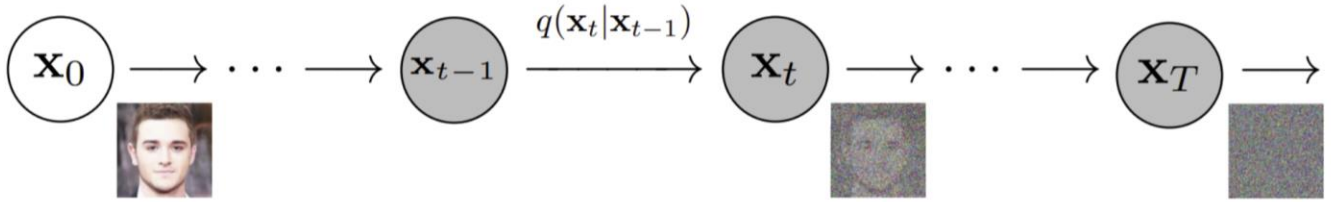
Diffusion models gradually add noise to data and learn to reverse this process



Co-funded by
the European Union

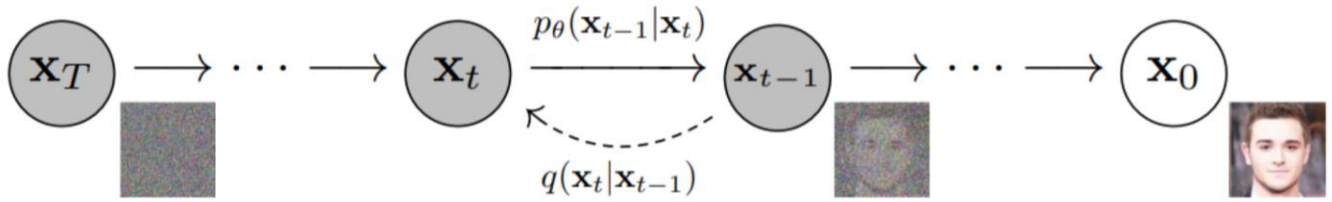
Diffusion Models

More specifically, a Diffusion Model is a latent variable model which maps to the latent space using a fixed Markov chain. This chain gradually adds noise to the data in order to obtain the approximate posterior $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$, where $\mathbf{x}_1, \ldots, \mathbf{x}_T$ are the latent variables with the same dimensionality as $\mathbf{x}_0$. In the figure below, we see such a Markov chain manifested for image data.



(Modified from source)

Ultimately, the image is asymptotically transformed to pure Gaussian noise. The **goal** of training a diffusion model is to learn the **reverse** process - i.e. training $p_\theta(x_{t-1}|x_t)$. By traversing backwards along this chain, we can generate new data.



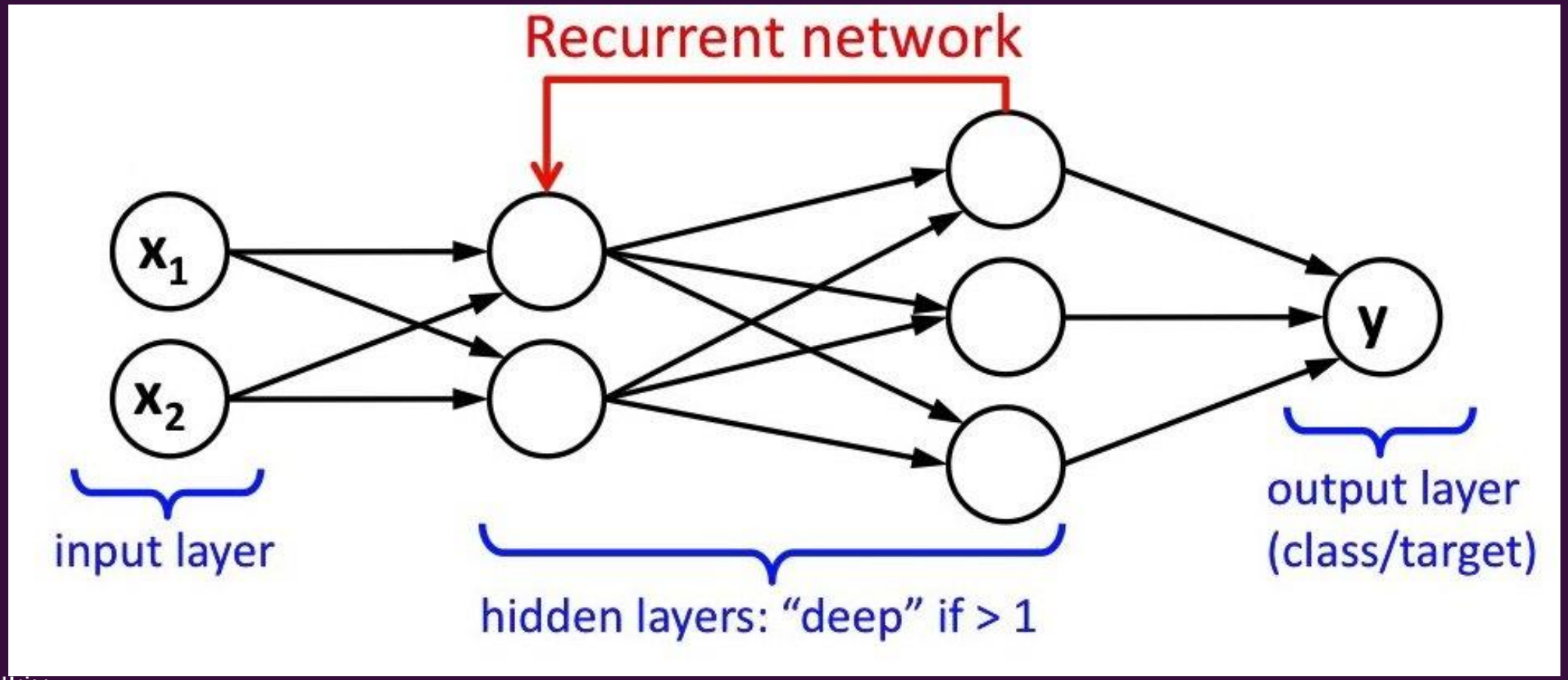
(Modified from source)

Co-funded by the European Union

https://www.assemblyai.com/blog/diffusion-models-for-machine-learning-introduction/

Recurrent Neural Networks

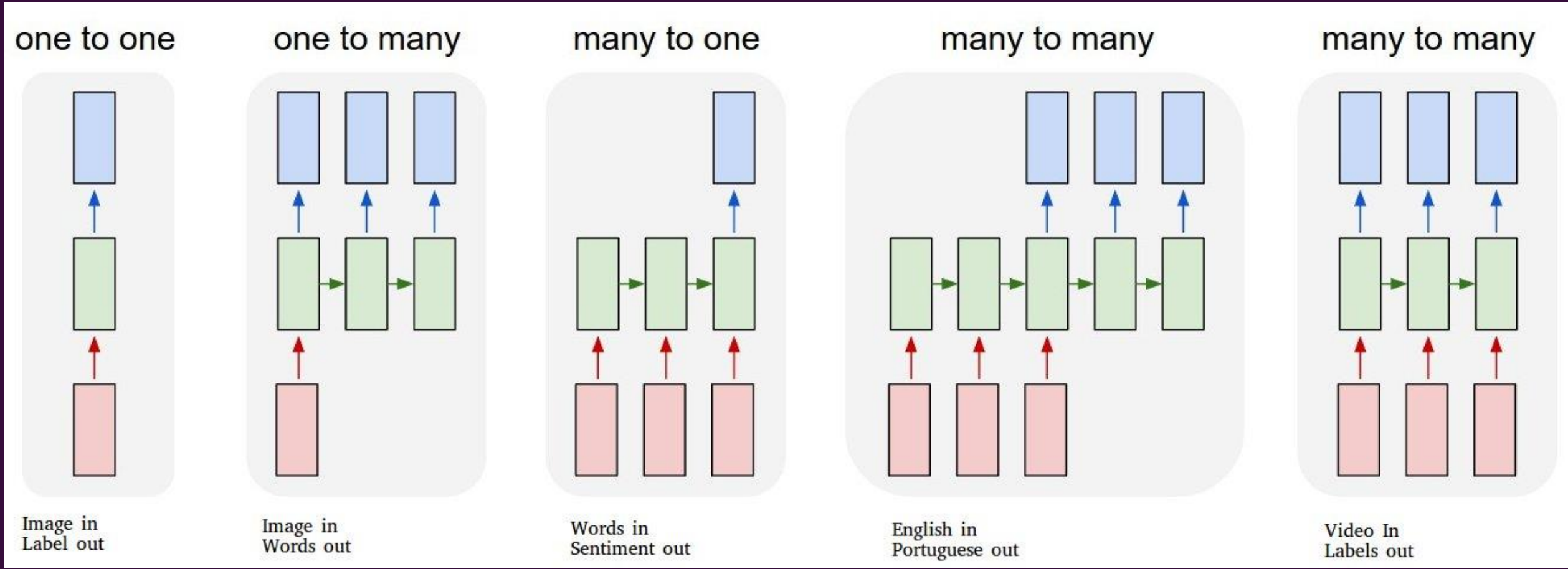
RNNs are designed for sequential data, such as time series and natural language, by maintaining memory across steps.



Co-funded by
the European Union

Recurrent Neural Networks

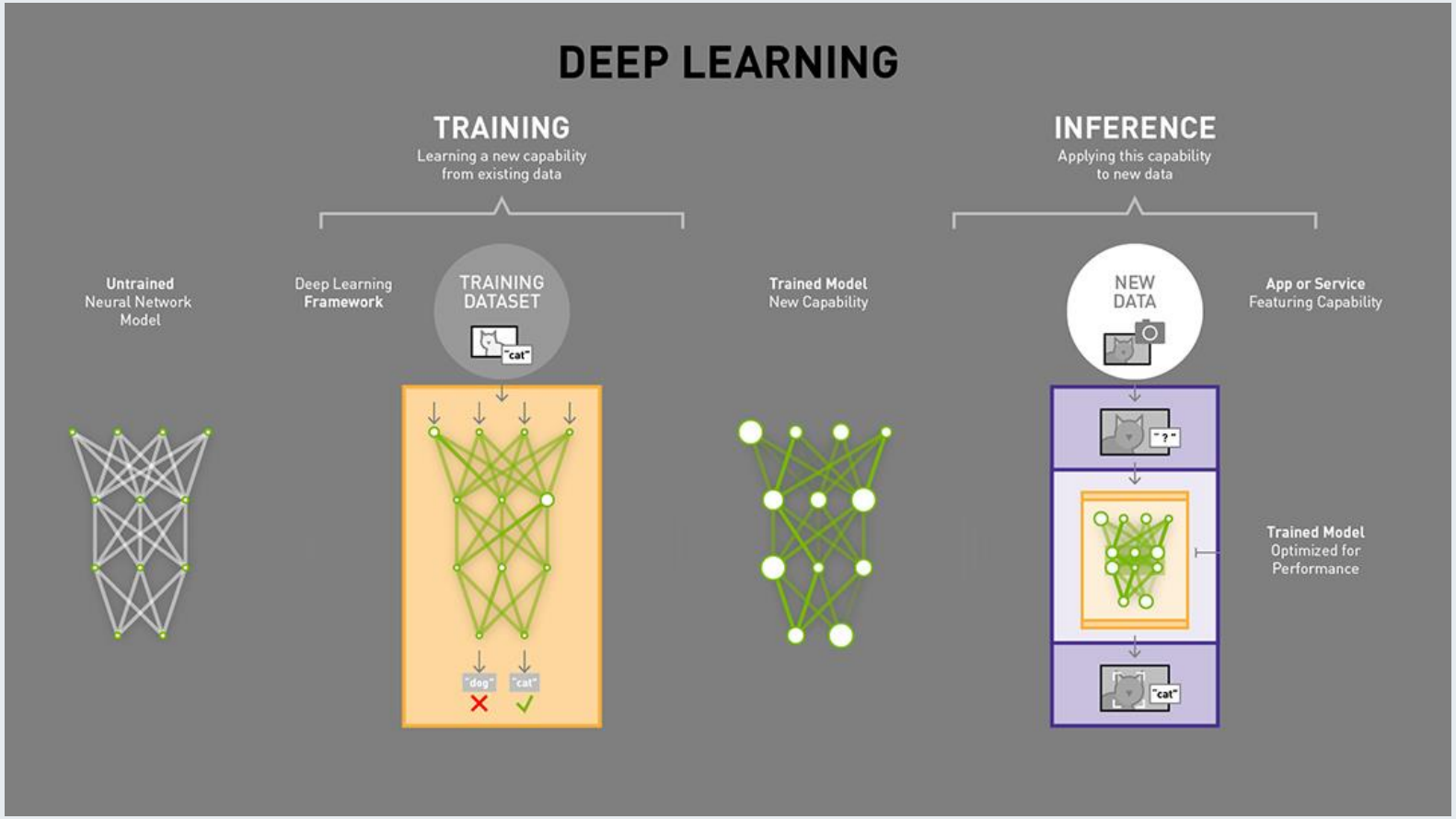
Different RNN architectures suit different types of sequence processing tasks



Co-funded by the European Union

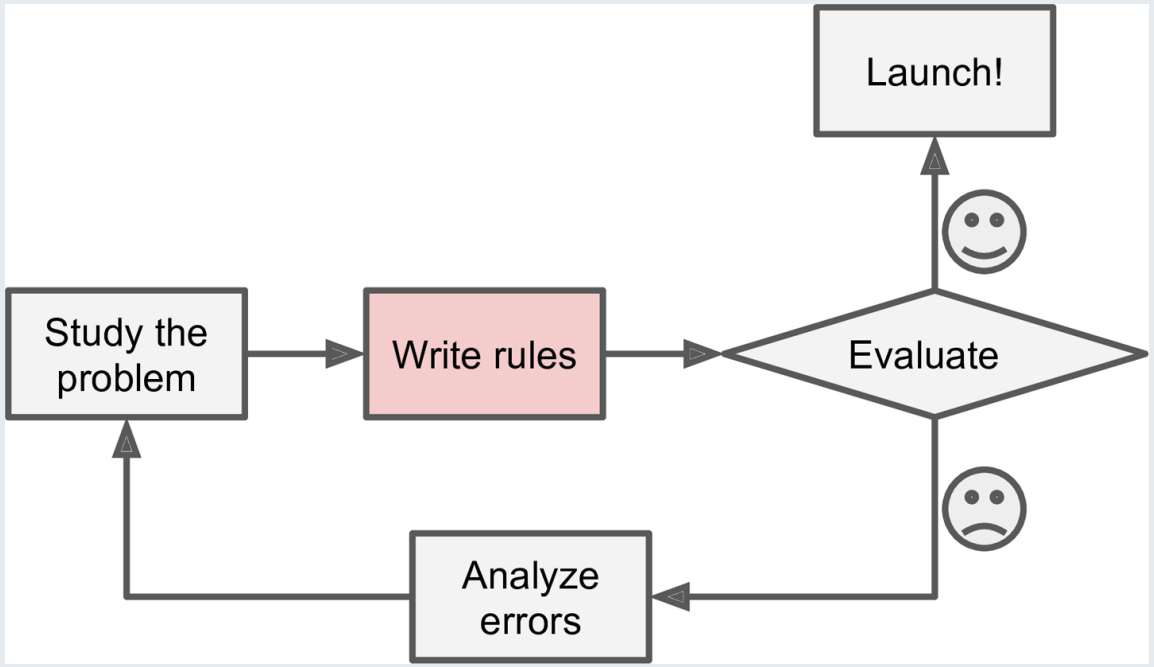
https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/recurrent_neural_networks.html

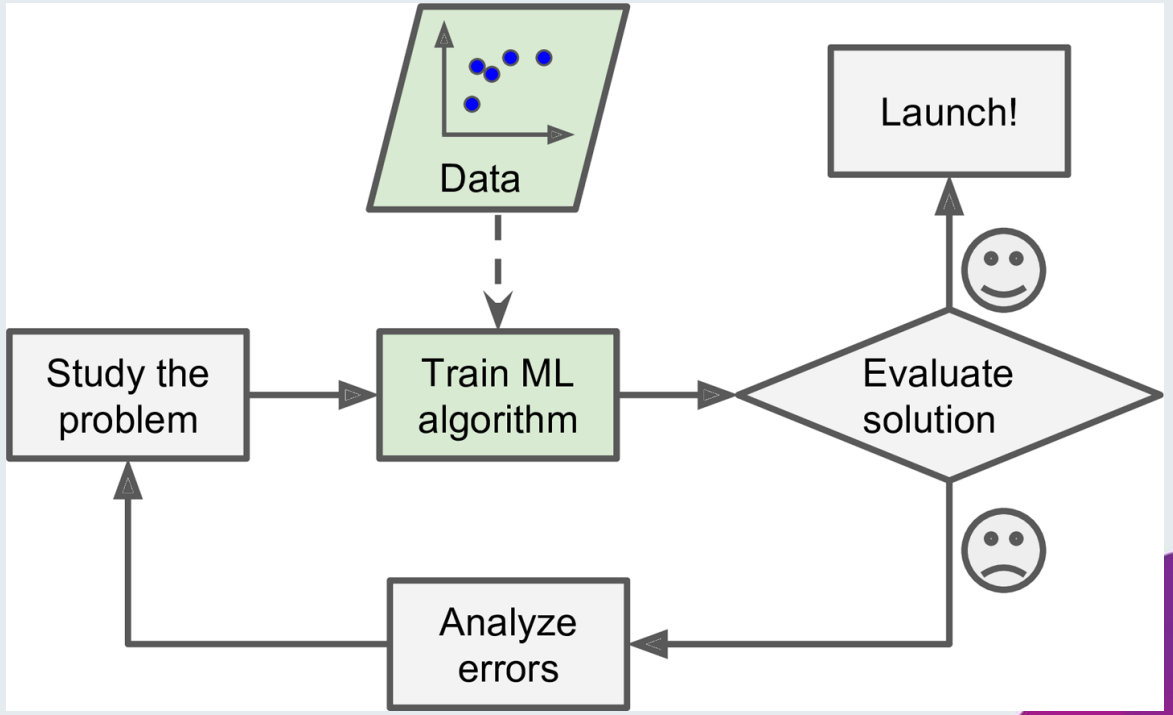
https://blogs.nvidia.com/blog/2016/08/22/difference-deep-learning-training-inference-ai/



Co-funded by the European Union

Traditional vs ML problem solving





Co-funded by the European Union

Neural Networks Limitations

- Very **data hungry** (eg. often millions of examples)
- **Computationally intensive** to train and deploy (tractably requires GPUs)
- Easily fooled by **adversarial examples**
- Can be subject to **algorithmic bias**
- Poor at **representing uncertainty** (how do you know what the model knows?)
- Uninterpretable **black boxes**, difficult to trust
- Often require **expert knowledge** to design, fine tune architectures
- Difficult to **encode structure** and prior knowledge during learning
- **Extrapolation**: struggle to go beyond the data

Co-funded by the European Union