# Generative Adversarial Networks (GANs)

## A gentle Introduction

https://github.com/toelt-llc/GANs-TensorFlow

Umberto Michelucci

umberto.michelucci@toelt.ai

+41 79 396 7406

# Who am I

- Head of AI Center of Excellence @ Helsana Versicherung AG
- Master in Theoretical Physics and PhD in Computer Science and Machine Learning
- Author of «Applied Deep Learning –A Case-Based Approach to Understanding Deep Neural Networks» (APRESS 2018)
- Author of «Advanced Applied Deep Learning – Convolutional Neural Networks and Object Detection» (APRESS 2019)
- Founder Toelt GmbH
- Google Developer Expert in Machine Learning
- More than 25 published papers in the last three years in Machine Learning

Helsana

Apress THE EXPERT'S VOICE

Apress THE EXPERT'S VOICE

TOELT.AI

Google

**Painter**: she would like to learn to learn to paint as Van Gogh

**Critic**: she would like to learn be able to tell real Van Gogh paintings from fakes apart.

# Step 1

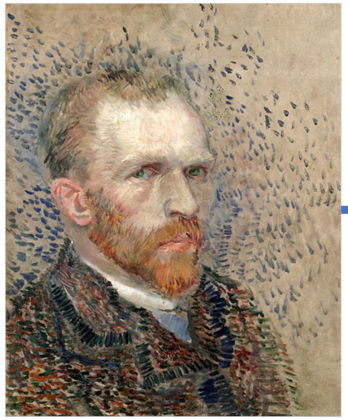
The painter becomes slowly better

Then the critic tells the painter what was clearly wrong.

**Critic**: she would like to learn be able to tell real Van Gogh paintings from fakes apart.

**Painter**: she would like to learn to learn to paint as Van Gogh

Produce a fake portrait that is not perfect



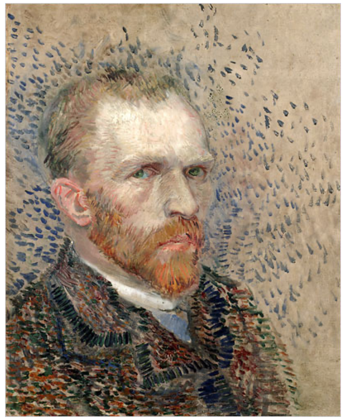
FAKE

The critic try to determine if the painting is fake or not.

# Step 2 – The critic must become better too

TRUE

The critic trains with real Van Gogh paintings



FAKE

The critic trains with **<u>clearly</u>** fake Van Gogh paintings

**Critic**: she would like to learn be able to tell real Van Gogh paintings from fakes apart.

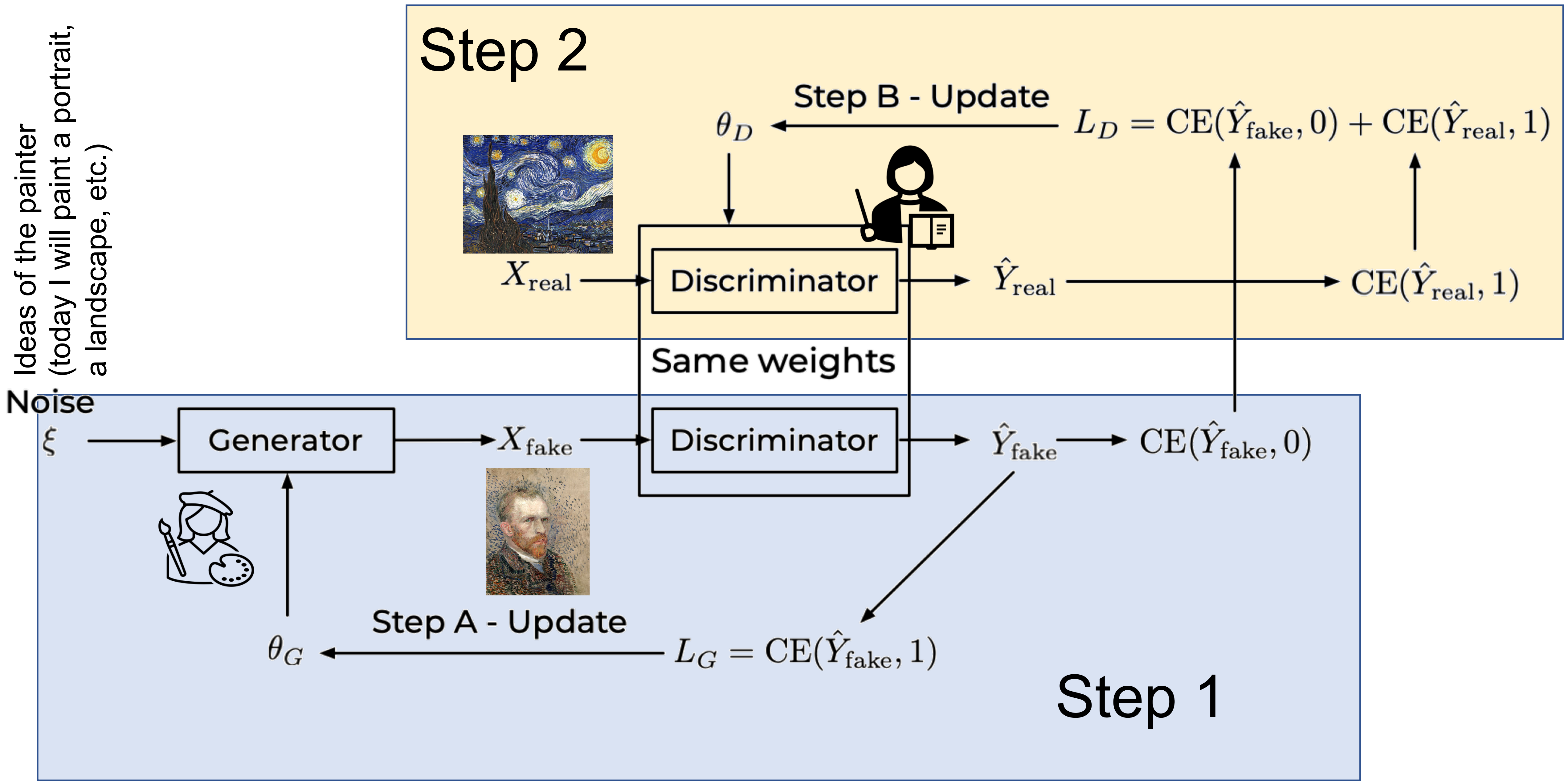
The critic becomes slowly better

# Reference – Loss Function (Cross Entropy, CE)

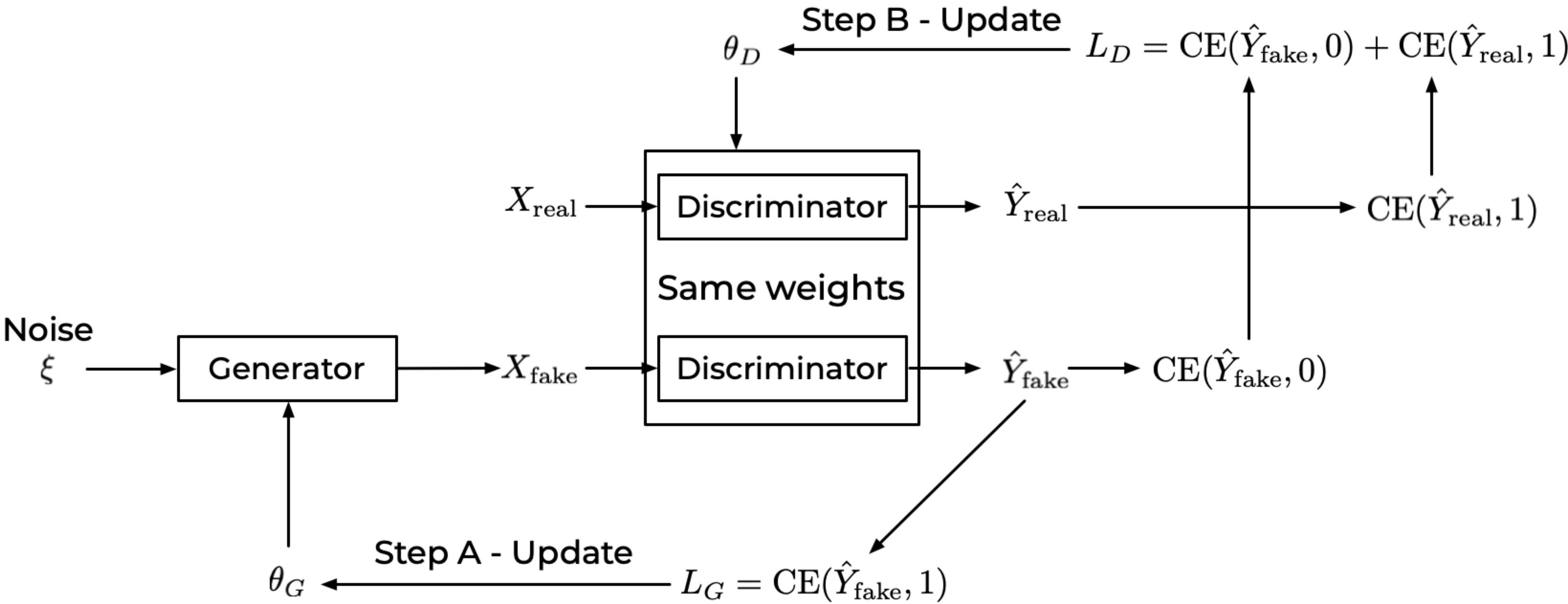
Predicted Class probability of painting of being real (class 1)

$$y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

Class Label: 0,1
0 → Painting is fake
1 → Painting is real



Step 2
Step B - Update
$\theta_D$
$L_D = \mathrm{CE}(\hat{Y}_{\text{fake}}, 0) + \mathrm{CE}(\hat{Y}_{\text{real}}, 1)$
$X_{\text{real}}$
Discriminator
$\hat{Y}_{\text{real}}$
$\mathrm{CE}(\hat{Y}_{\text{real}}, 1)$
Same weights
Ideas of the painter
(today I will paint a portrait,
a landscape, etc.)
Noise
$\xi$
Generator
$X_{\text{fake}}$
Discriminator
$\hat{Y}_{\text{fake}}$
$\mathrm{CE}(\hat{Y}_{\text{fake}}, 0)$
Step A - Update
$\theta_G$
$L_G = \mathrm{CE}(\hat{Y}_{\text{fake}}, 1)$
Step 1


Step B - Update
$\theta_D$
$L_D = \mathrm{CE}(\hat{Y}_{\text{fake}}, 0) + \mathrm{CE}(\hat{Y}_{\text{real}}, 1)$
$X_{\text{real}}$
Discriminator
$\hat{Y}_{\text{real}}$
$\mathrm{CE}(\hat{Y}_{\text{real}}, 1)$
Same weights
Noise
$\xi$
Generator
$X_{\text{fake}}$
Discriminator
$\hat{Y}_{\text{fake}}$
$\mathrm{CE}(\hat{Y}_{\text{fake}}, 0)$
Step A - Update
$\theta_G$
$L_G = \mathrm{CE}(\hat{Y}_{\text{fake}}, 1)$

# How to use Keras for GANs

The most important aspect of using Keras for GANs is the necessity of building a custom training loop (and not using **compile()/fit()** approach.

```python
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
      # Calculation of X_{fake}
      generated_images = generator(noise, training=True)

      # Calculation of \hat Y_{real}
      real_output = discriminator(images, training=True)

      # Calculation of \hat Y_{fake}
      fake_output = discriminator(generated_images, training=True)

      # Calculation of L_G
      gen_loss = generator_loss(fake_output)

      # Calculation of L_D
      disc_loss = discriminator_loss(real_output, fake_output)

    #
    # Gradients Calculation
    #
    # Calculation of the gradients of L_G for backpropagation
    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)

    # Calculation of the gradients of L_D for backpropagation
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

    #
    # Training Steps A and B
    #
    # Step A
    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))

    # Step B
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.trainable_variables))
```

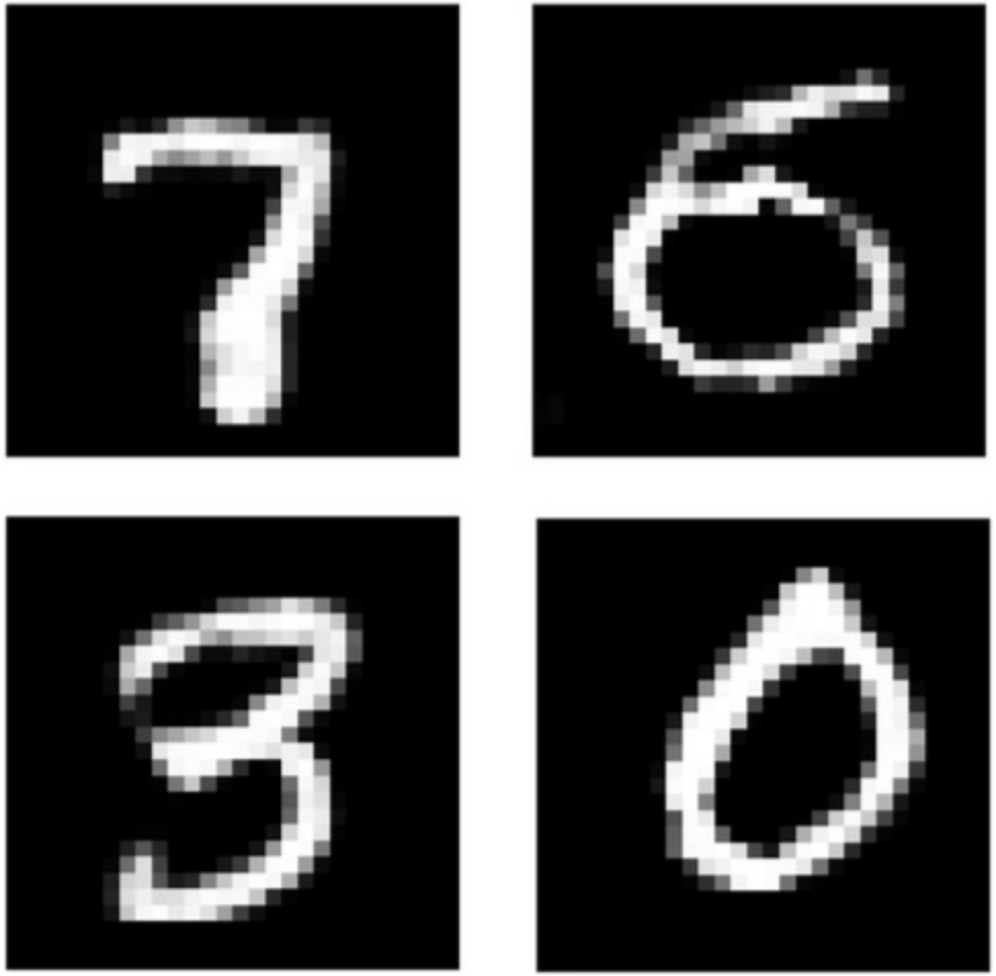
$X_{\text{fake}}$

$X_{\text{real}}$

$$L_G = \mathrm{CE}(\hat{Y}_{\text{fake}}, 1)$$

$$L_D = \mathrm{CE}(\hat{Y}_{\text{fake}}, 0) + \mathrm{CE}(\hat{Y}_{\text{real}}, 1)$$

# The digits were not written by a person…

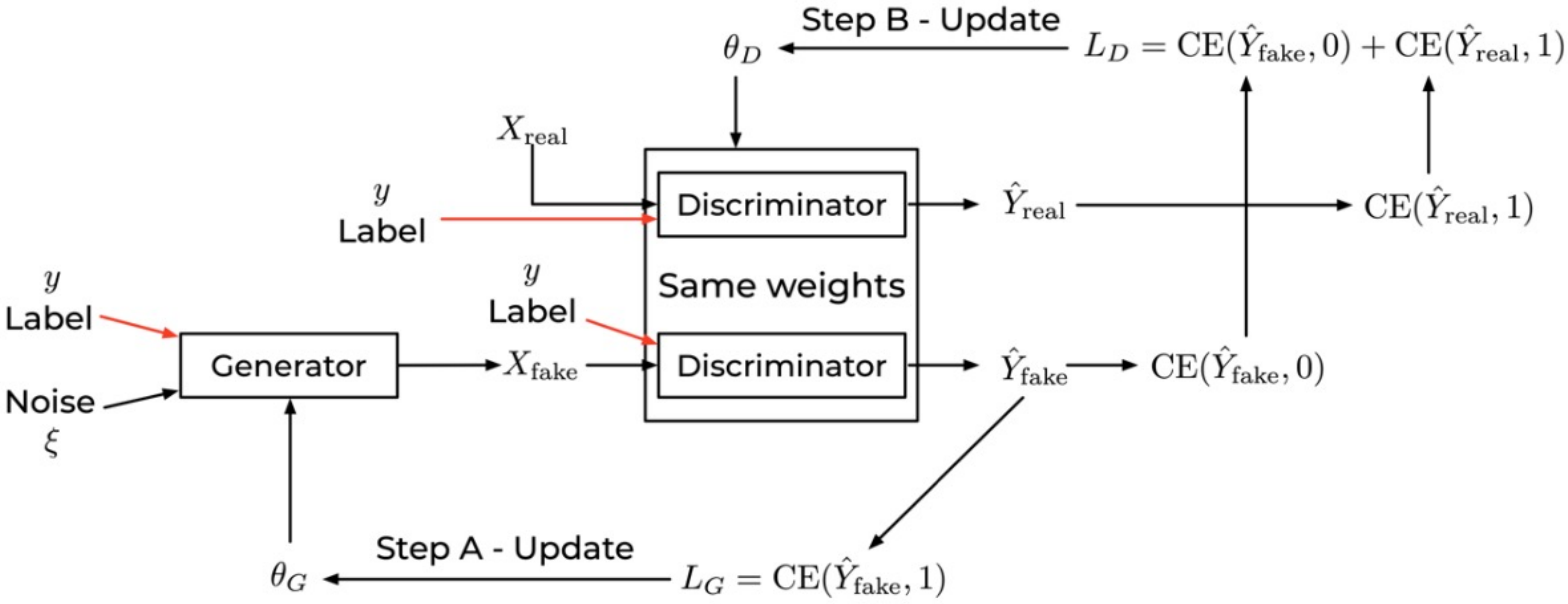


# Conditional GAN

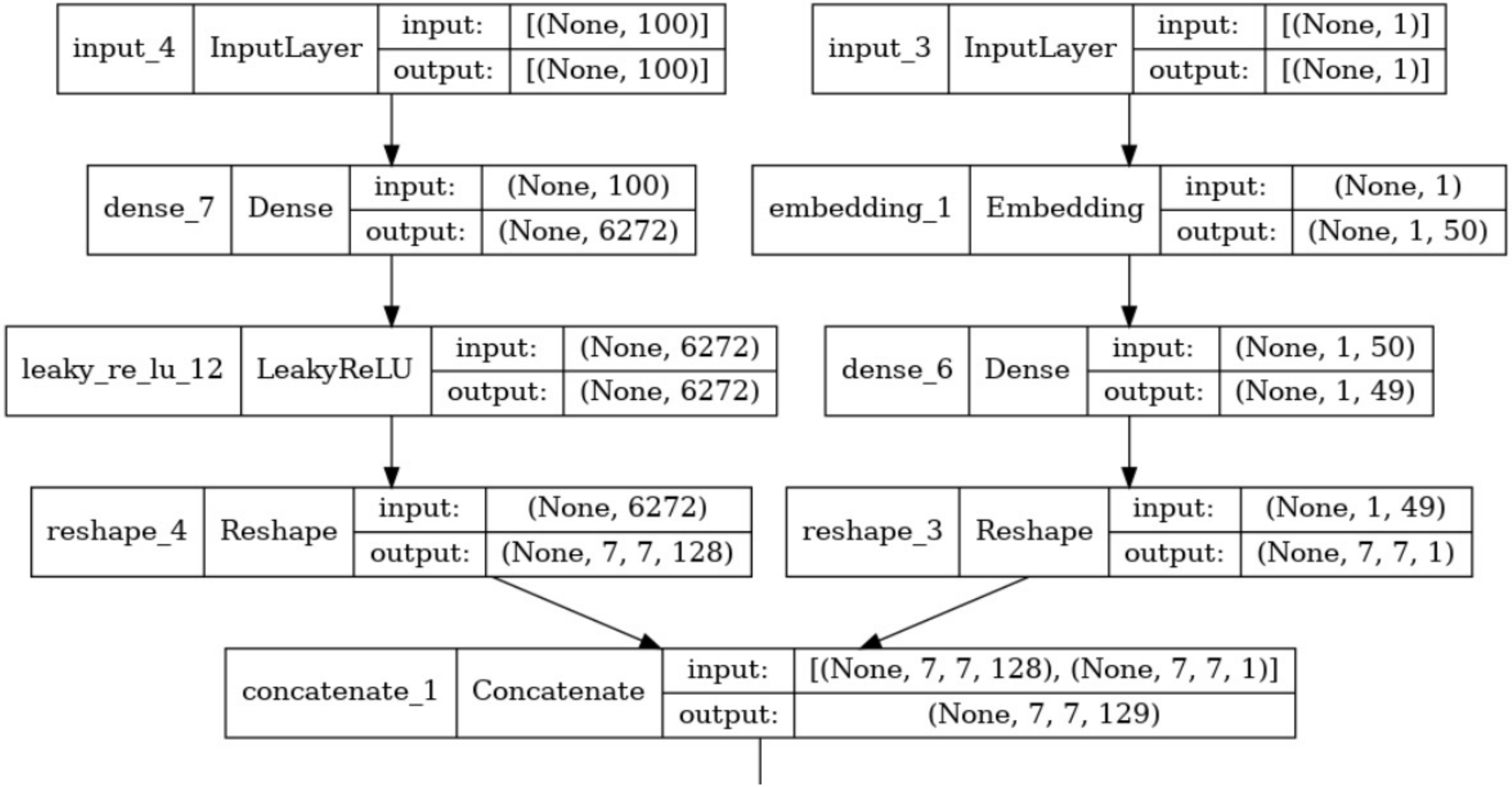
Conditional GAN (CGAN) is **a GAN variant in which both the Generator and the Discriminator are conditioned on auxiliary data such as a class label during training**.

Source: https://livebook.manning.com/book/gans-in-action/chapter-8/

# Conditional GAN



# First Layers of the generator (CGAN)



# First Layers of the discriminator (CGAN)