DEEP LEARNING ADVENTURE

By : Ahmed Haytham

GANS AND DIFFUSION MODELS

Introduction

Greetings, fellow adventurer! My name is Ahmed Haytham, and I have authored this booklet by compiling my personal notes from a range of courses and profound books within the fascinating domain of Deep Learning. If you wish to connect and engage with me further, I can be found on LinkedIn, eagerly awaiting the opportunity to communicate with you.

For Who ?

If this is your first encounter with these topics or if you're in need of a quick recap

must be familiar with CNN & image classfication

What are GANS ?

A generative adversarial network (GAN) is a machine learning model that learns to generate new data that is similar to a set of existing data. GANs are trained using a technique called "adversarial learning," in which two neural networks compete with each other.

introduced by Ian Goodfellow and other researchers at the University of Montreal, including Yoshua Bengio, in 2014.

GANs are just mechanisms to create new samples; they do not build a probability distribution over the modeled data and hence cannot evaluate the probability that a new data point belongs to the same distribution. GANs have been applied to many types of data, including audio, 3D models, text, video, and graphs.

GAN architecture

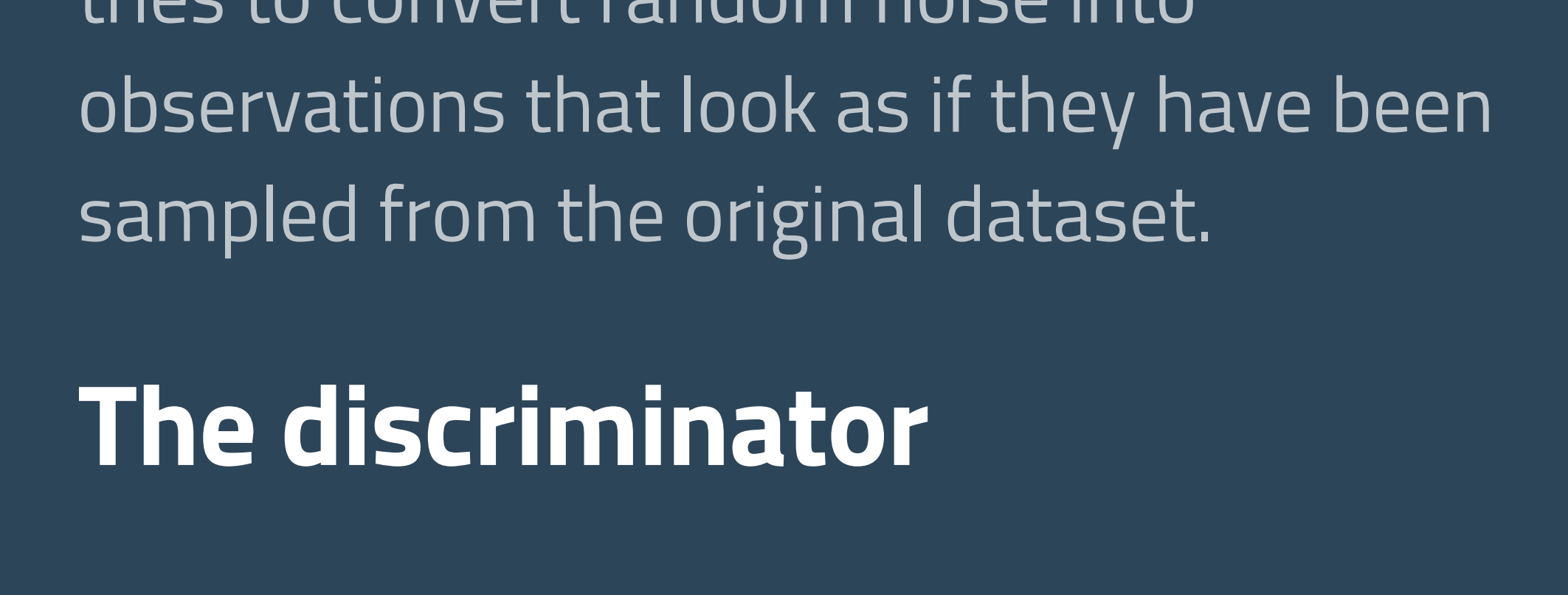
GANs are based on the idea of adversarial training. The GAN architecture basically consists of two neural networks that compete against each other:

The generator

tries to convert random noise into observations that look as if they have been sampled from the original dataset.

The discriminator

tries to predict whether an observation comes from the original dataset or is one of the generator's forgeries.

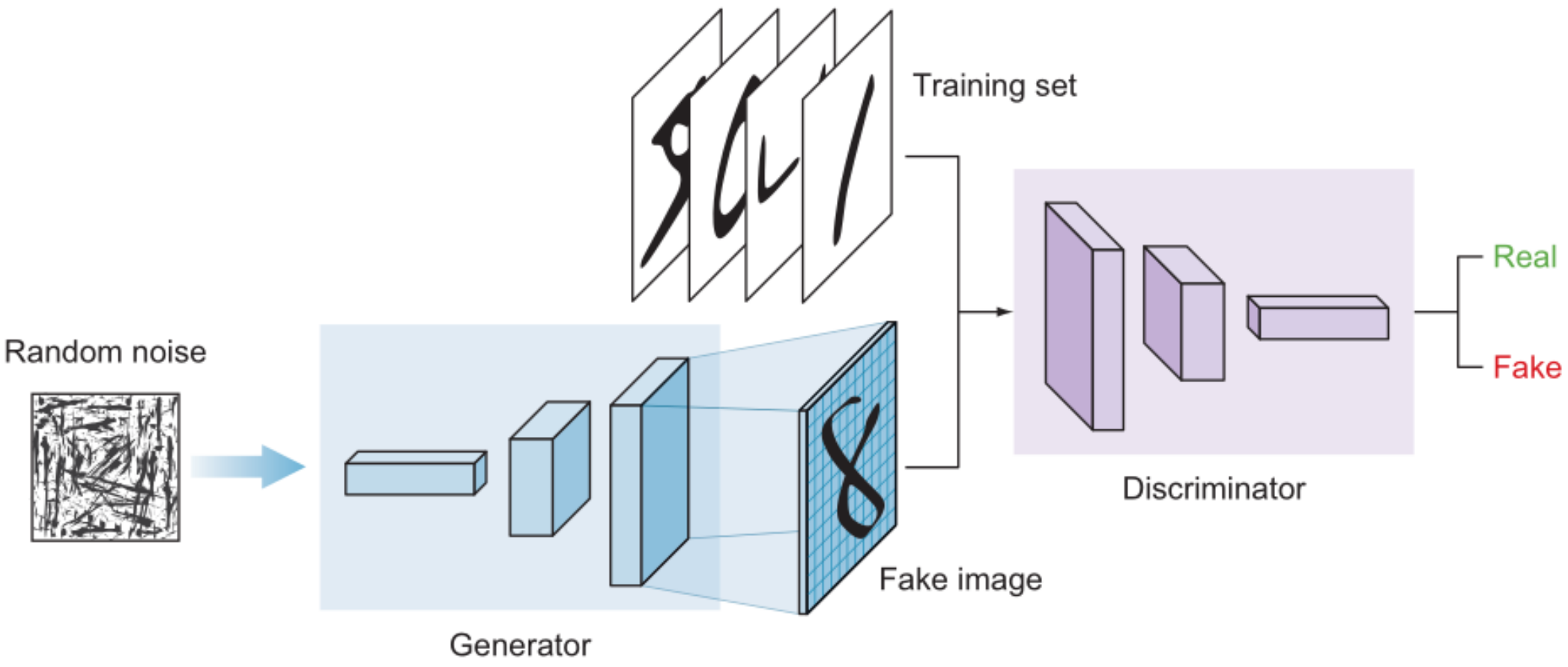


Dear fellow adventurer, let me shed light on the fascinating world of Generative Adversarial Networks (GANs). In a GAN, there are two distinct models working together: the generator and the discriminator. These two models operate simultaneously, engaging in a training process akin to a zero-sum game, where each model strives to outperform the other. It's an enthralling dynamic that unfolds as they battle for supremacy.

(DCGANs)

In the original GAN paper in 2014, multilayer perceptron (MLP) networks were used to build the generator and discriminator networks.

The deep convolutional GAN or DCGAN was an early GAN architecture specialized for generating images

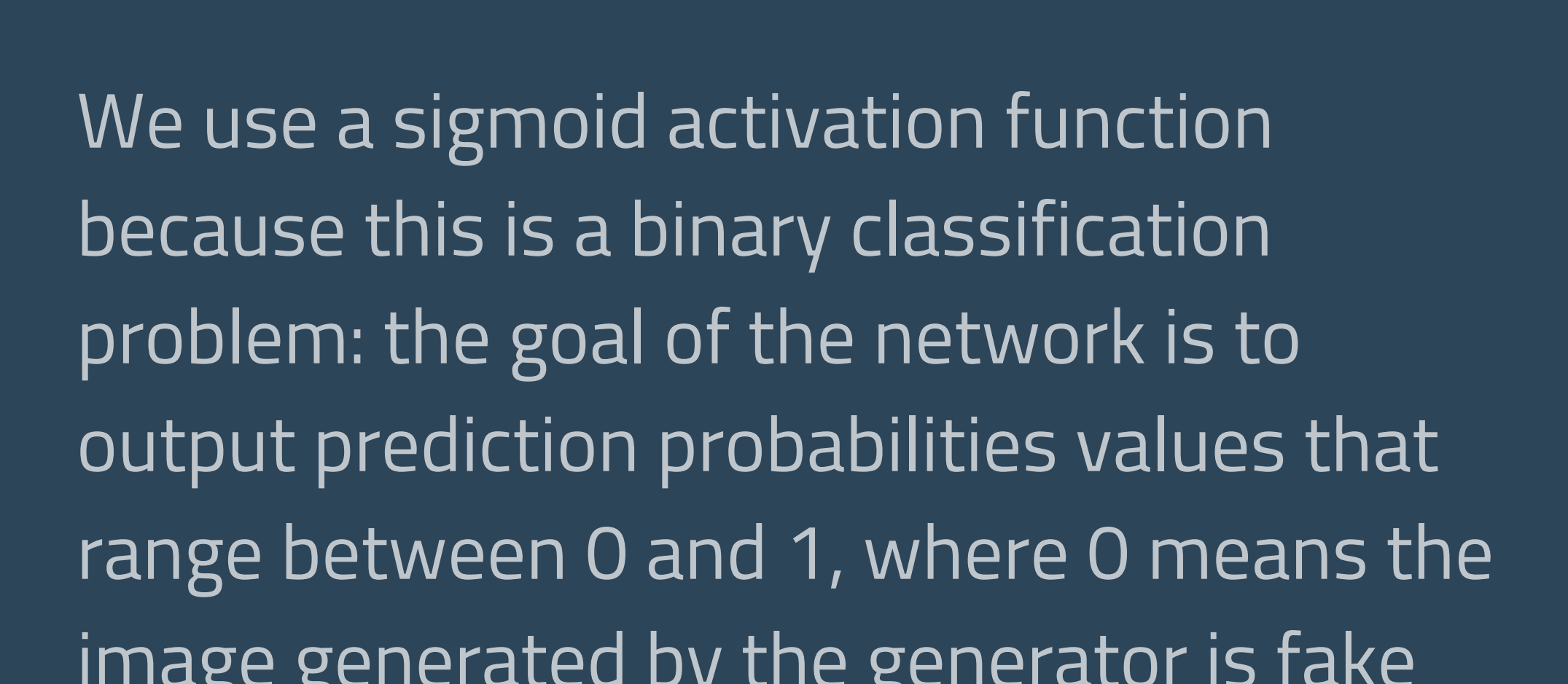


The C.discriminator model

As explained earlier, the goal of the discriminator is to predict whether an image is real or fake

This is a typical supervised classification problem

We use a sigmoid activation function because this is a binary classification problem: the goal of the network is to output prediction probabilities values that range between 0 and 1, where 0 means the image generated by the generator is fake and 1 means it is 100% real.

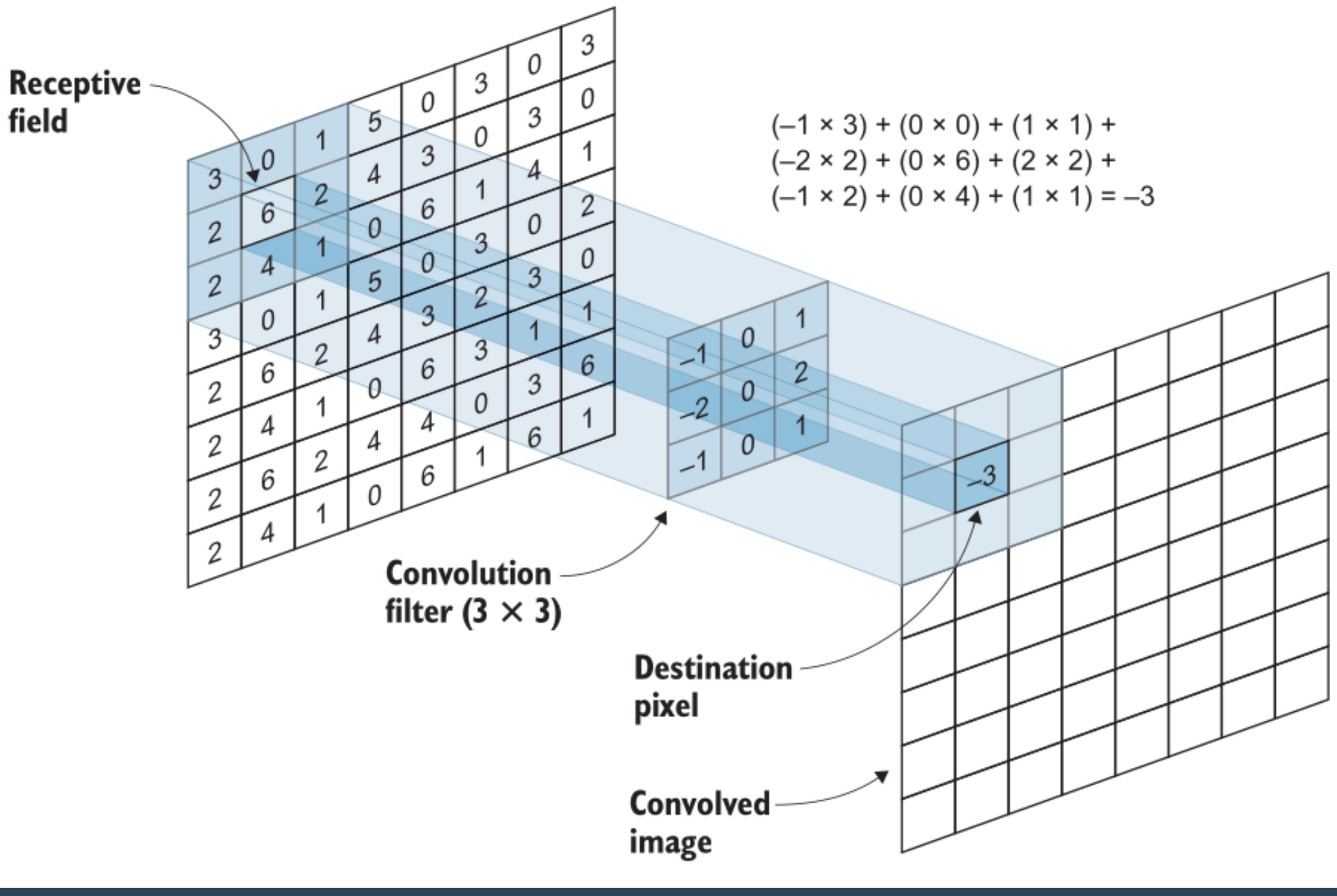


Convolutional layers

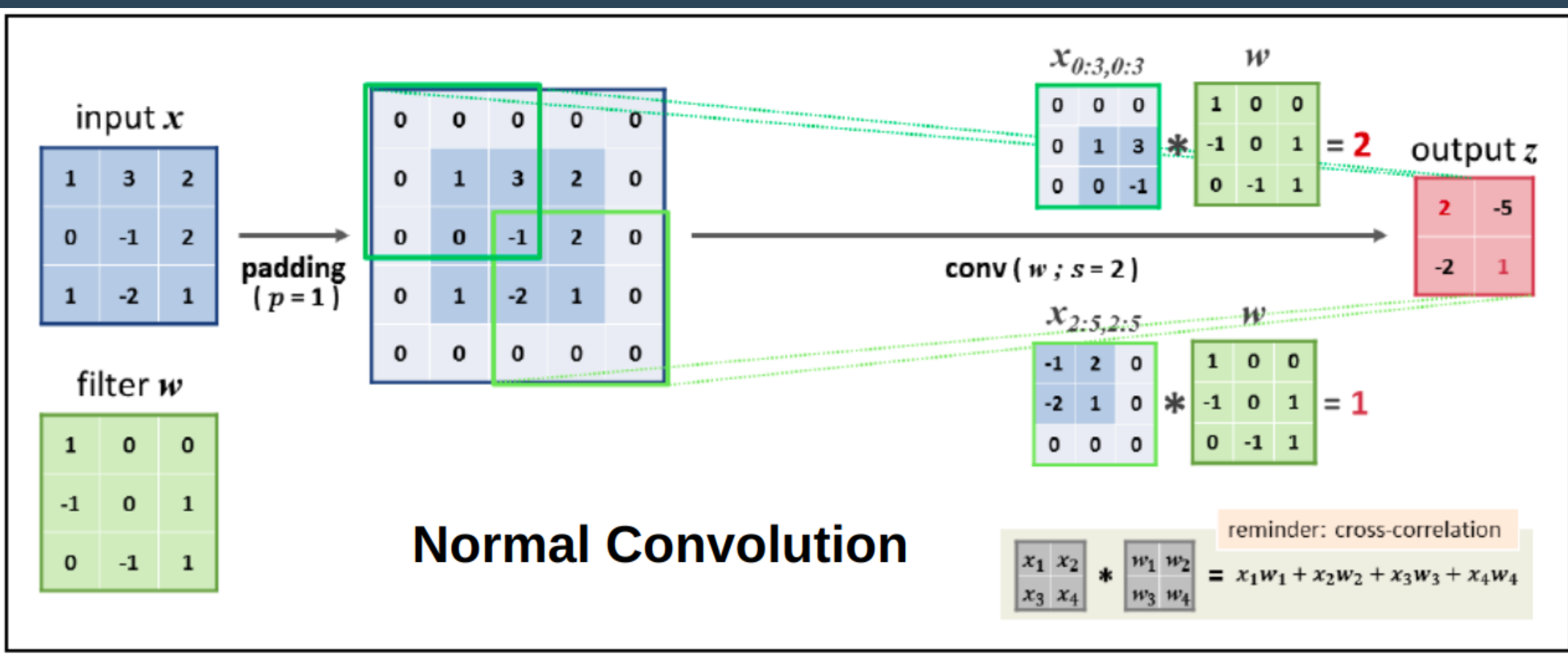
Dear adventurer, this is a quick recap on convolution layers because we will soon be covering transposed convolution layers, which are sometimes mistakenly called deconvolution layers.

A convolutional layer is the core building block of a convolutional neural network. Convolutional layers act like a feature finder window that slides over the image pixel by pixel to extract meaningful features that identify the objects in the image.

In mathematics, convolution is the operation of two functions to produce a third modified function



the first function is the input image, and the second function is the convolutional filter. We will perform some mathematical operations to produce a modified image with new pixel values.

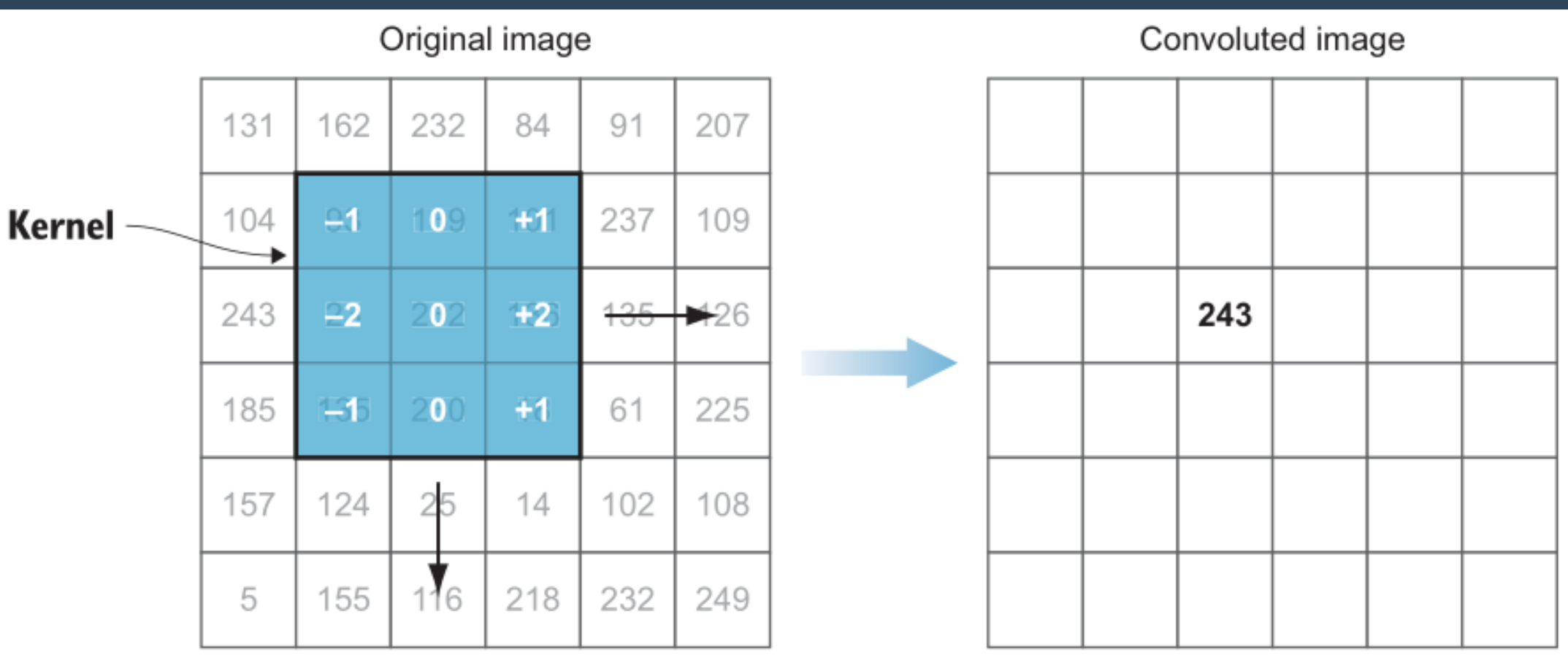


Convolution kernel
with optimized weights

$$
* \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} =
$$

The math should look familiar from our discussion of MLPs. Remember how we multiplied the input by the weights and summed them all together to get the weighted sum?

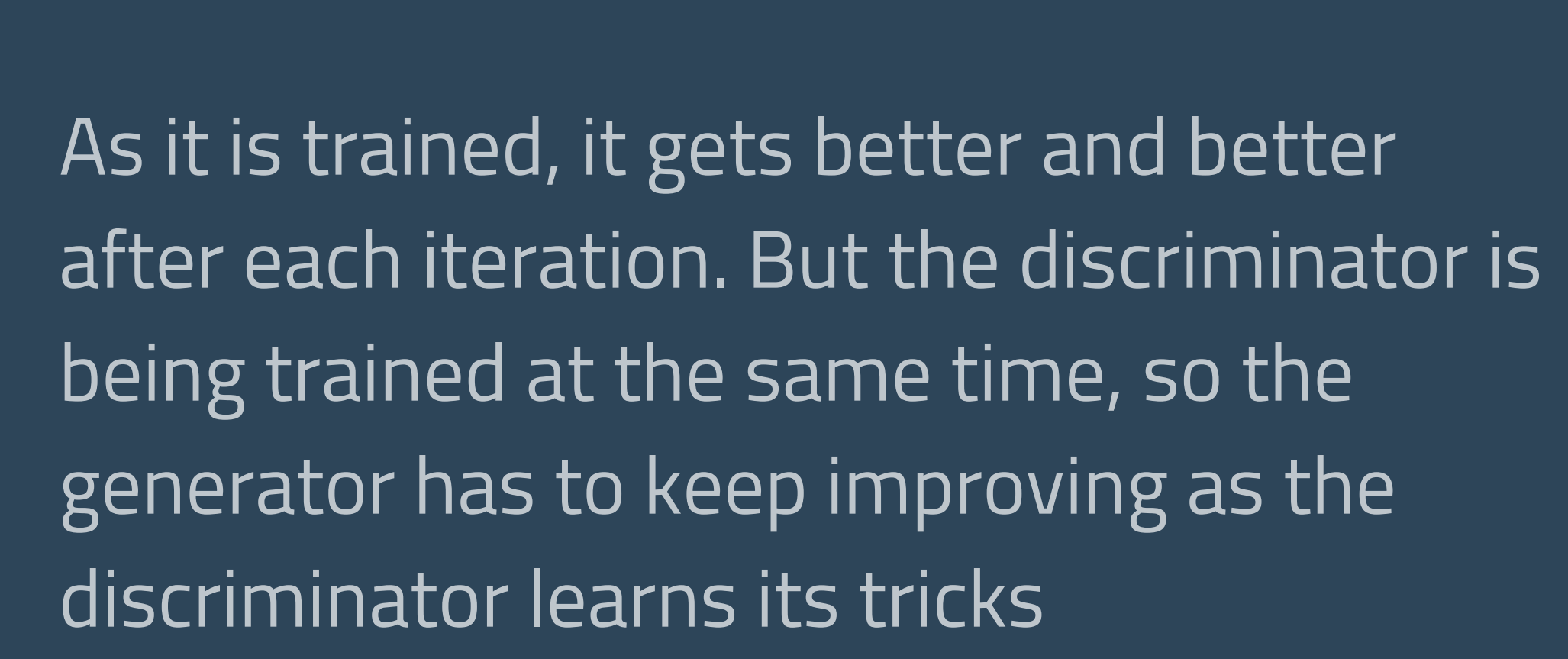
$$WB = x1 \cdot w1 + x2 \cdot w2 + x3 \cdot w3 + \ldots + xn \cdot wn + b$$



The C. generator model

The generator takes in some random data and tries to mimic the training dataset to generate fake images. Its goal is to trick the discriminator by trying to generate images that are perfect replicas of the training dataset

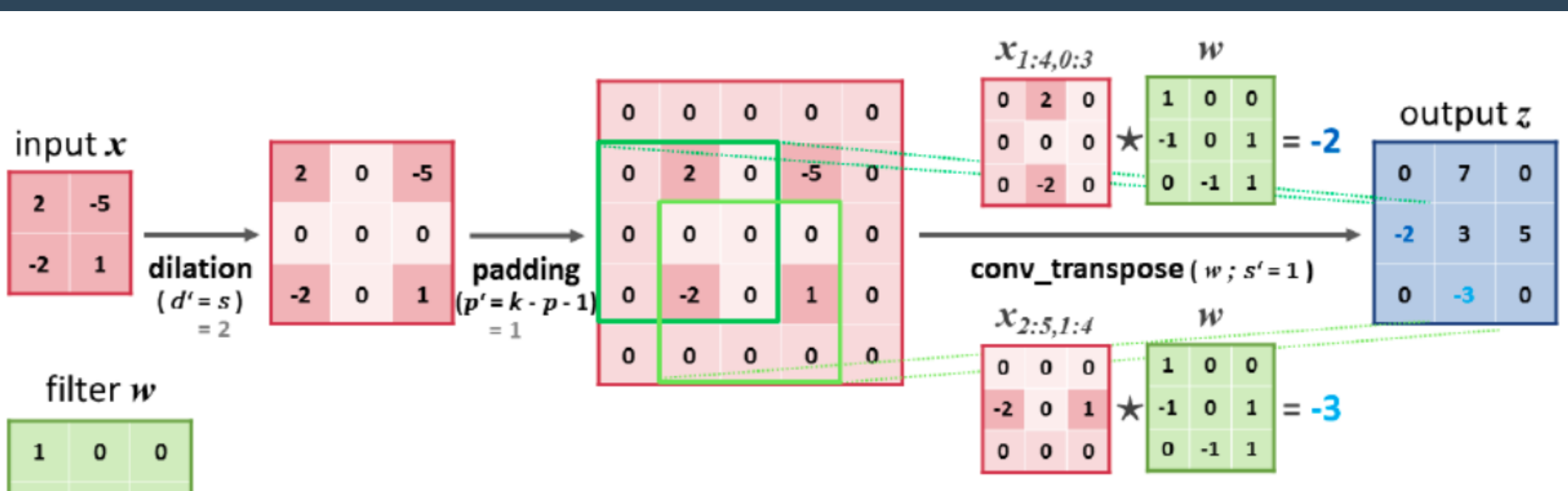
As it is trained, it gets better and better after each iteration. But the discriminator is being trained at the same time, so the generator has to keep improving as the discriminator learns its tricks



Transposed Convolution Upsampling

A transposed convolutional layer is typically used for upsampling, which means it can generate an output feature map with a larger size than the input feature map.

are a type of layer used in neural networks to increase the spatial resolution of feature maps.

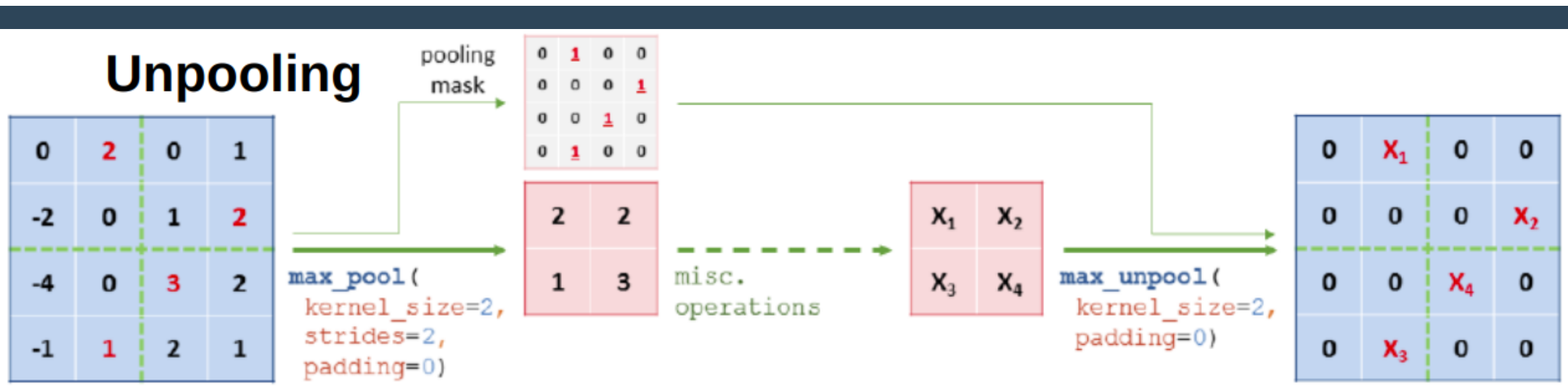


Transposed convolution

reminder: convolution (math)

$$\begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix} \star \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} = x_1 w_4 + x_2 w_3 + x_3 w_2 + x_4 w_1$$

Unpooling



Difficulty Training GANs

In theory, GANs are relatively simple to understand. However, in practice, they can be very difficult to train.

To train the DCGAN reliably, it was necessary to make the following choices:

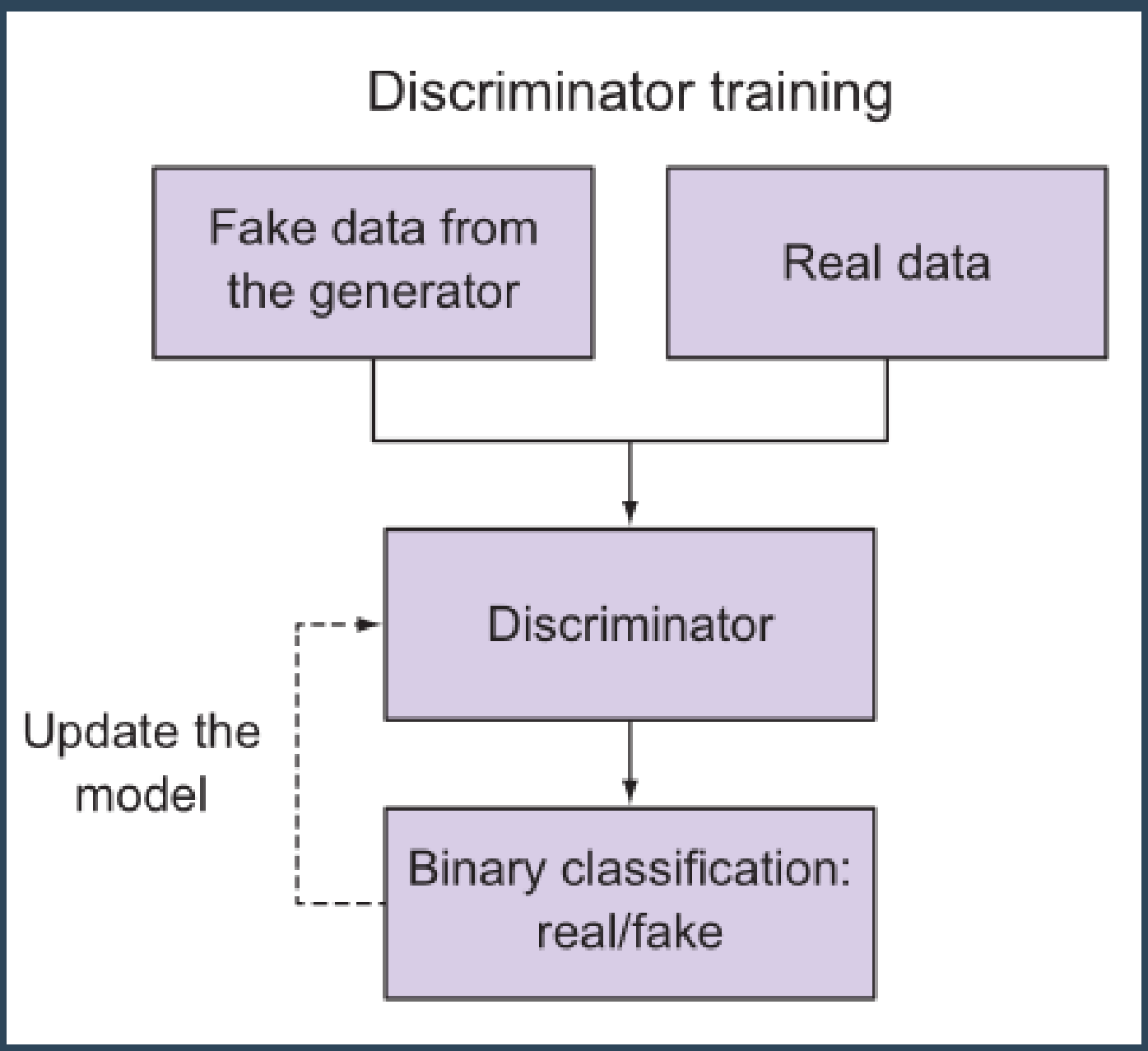
1. Use strided convolutions for upsampling and downsampling.
2. Use BatchNorm in both the generator and discriminator, except in the last and first layers.
3. Use the leaky ReLU activation function in the discriminator.
4. Use the Adam optimizer, but with a lower momentum coefficient than usual.

These choices are unusual because most deep learning models are relatively robust to such choices. However, GANs are notoriously difficult to train, and these choices help to stabilize the training process.

Now that we've learned the discriminator and generator models separately, let's put them together to train an end-to-end generative adversarial network

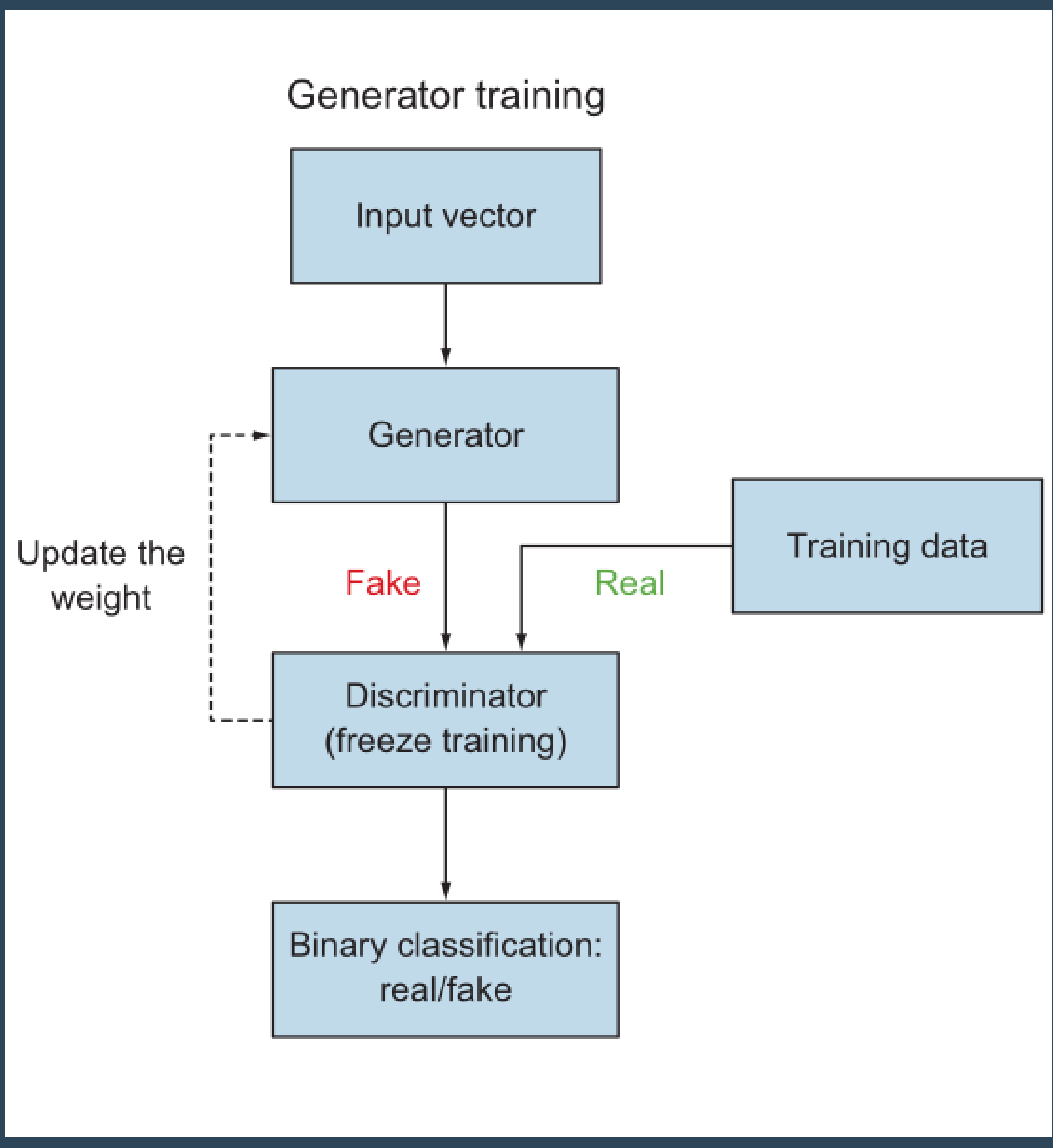
Train the discriminator

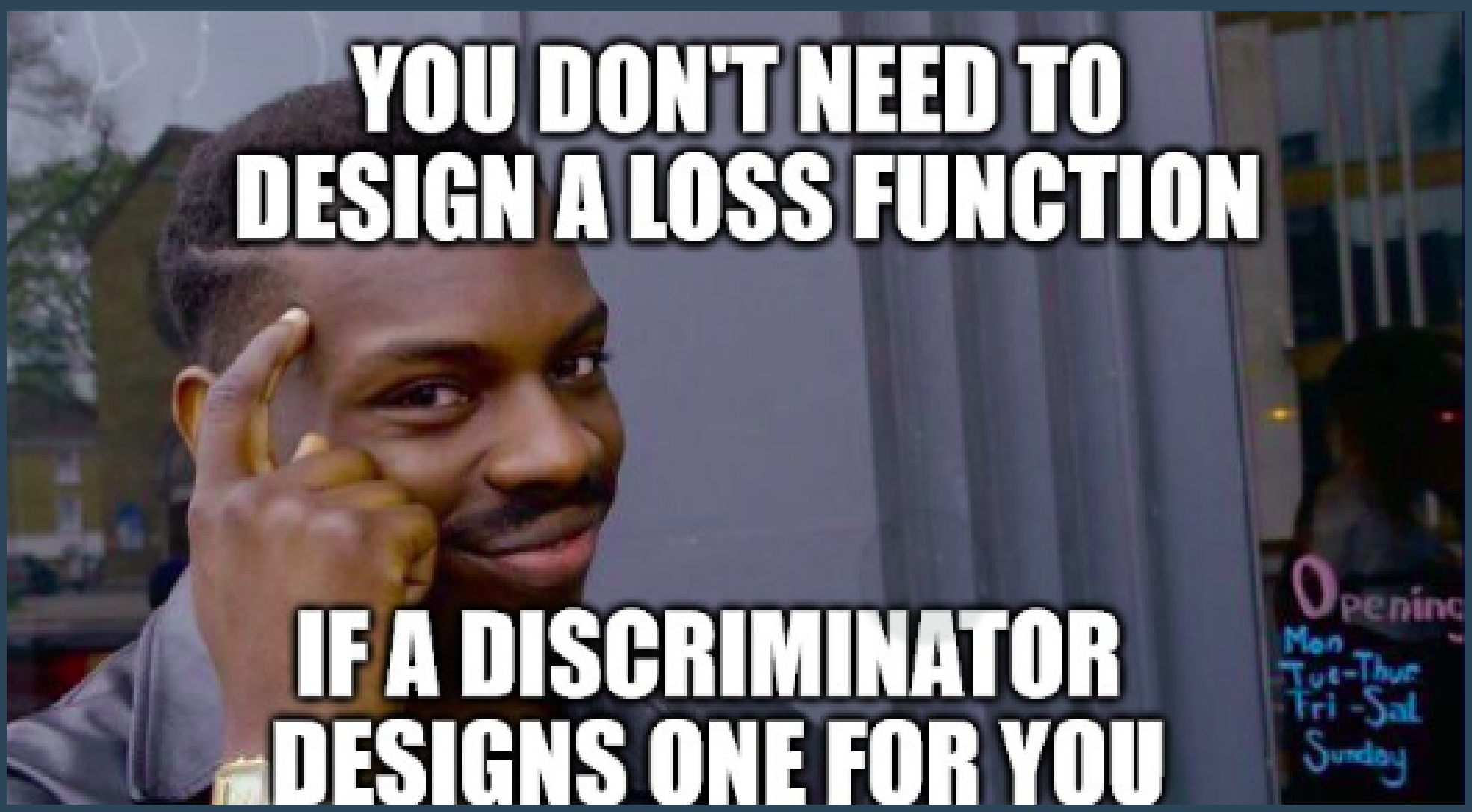
This is a straightforward supervised training process. The network is given labeled images coming from the generator (fake) and the training data (real), and it learns to classify between real and fake images with a sigmoid prediction output. Nothing new here.



Train the generator.

This process is a little tricky. The generator model cannot be trained alone like the discriminator. It needs the discriminator model to tell it whether it did a good job of faking images. So, we create a combined network to train the generator, composed of both discriminator and generator models.



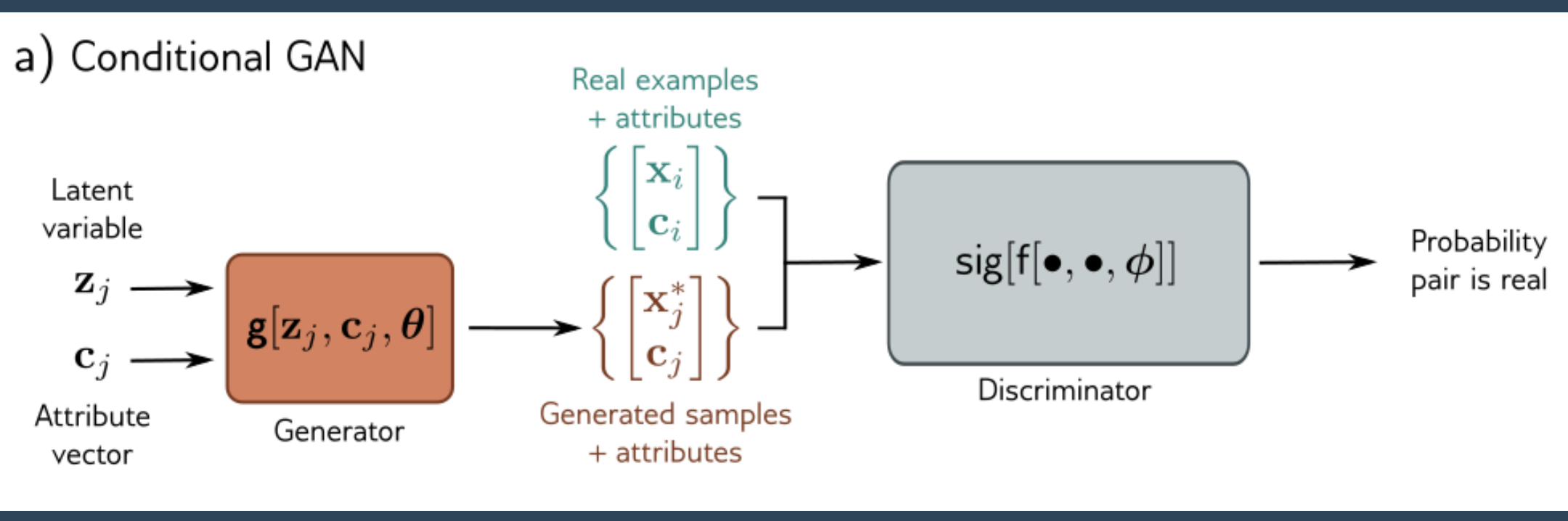


Conditional generation

Hey adventurer, did you know that GANs can produce realistic images? They can be used to create all sorts of things, like landscapes, characters, and even objects. But one thing that GANs can't do is specify the attributes of the images they create. For example, you can't tell a GAN to create a picture of a green-haired, Egyptian man. To do that, you would need to train a separate GAN for each combination of characteristics.

Conditional GAN

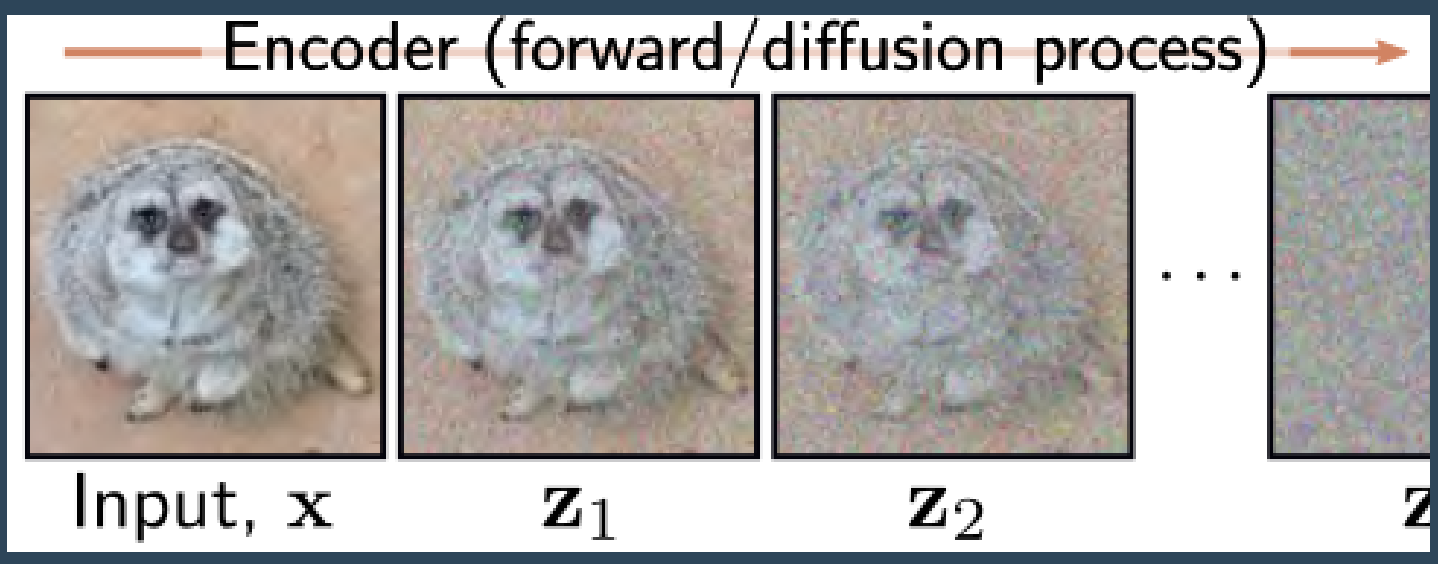
The conditional GAN passes a vector c of attributes to both the generator and discriminator, which are now written as g[z, c, θ] and f[x, c, ϕ], respectively. The generator aims to transform the latent variable z into a data sample x with the correct attribute c. The discriminator's goal is to distinguish between (i) the generated sample with the target attribute or (ii) a real example with the real attribute



for more, there are Auxiliary classifiers GAN CycleGAN and others

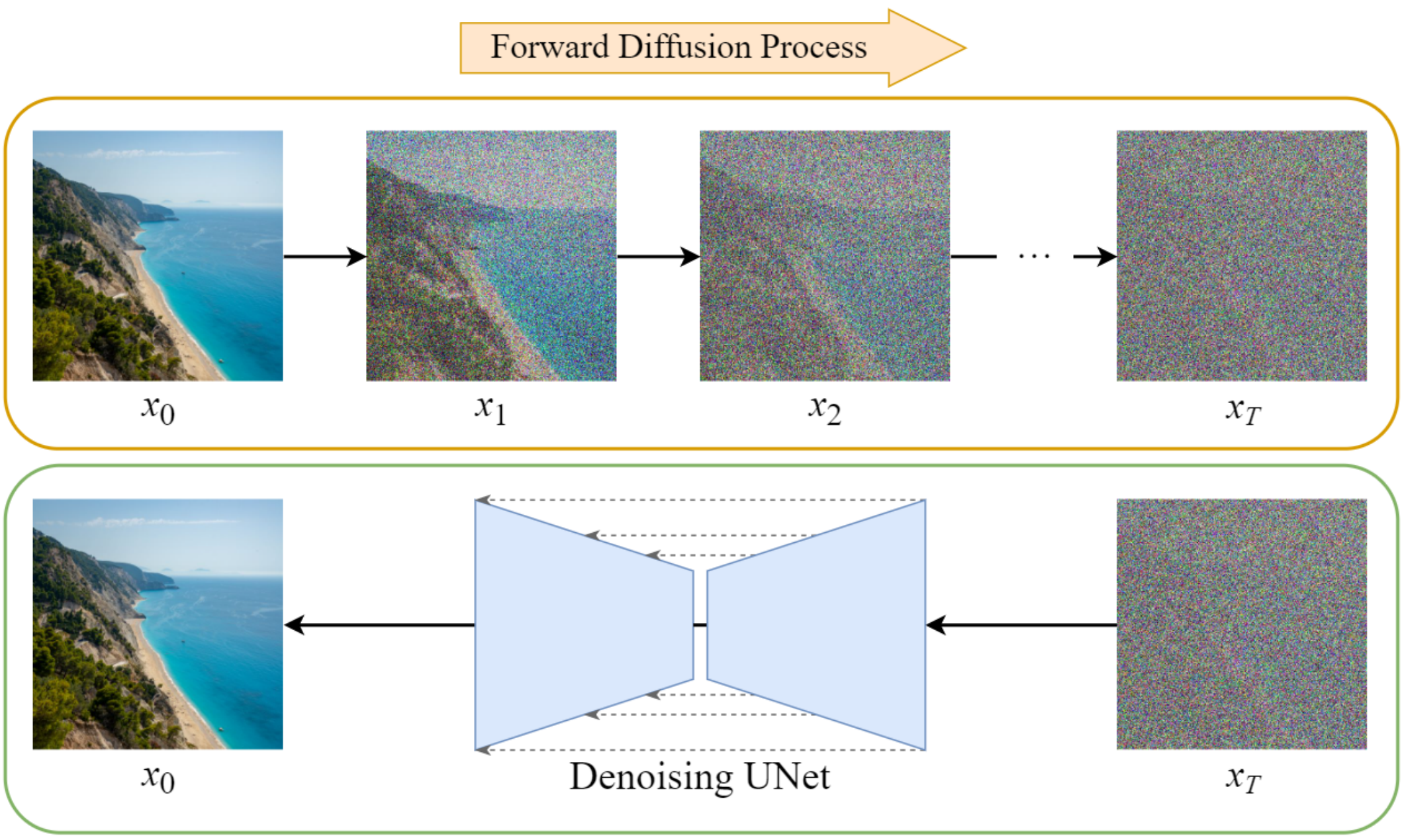
Diffusion models

These are probabilistic models that define a nonlinear mapping from latent variables to the observed data where both quantities have the same dimension. Like variational autoencoders, they approximate the data likelihood using a lower bound based on an encoder that maps to the latent variable.



Overview

A diffusion model consists of an encoder and a decoder. The encoder takes a data sample x and maps it through a series of intermediate latent variables z1 . . . zT . The decoder reverses this process; it starts with zT and maps back through zT −1 , . . . , z1 until it finally (re-)creates a data point x



Encoder (forward process)

The diffusion or forward process1 (figure 18.2) maps a data example x through a series
of intermediate variables z1 , z2 , . . . , zT with the same size as x according to

$$\begin{aligned}
\mathbf{z}_1 &= \sqrt{1-\beta_1}\cdot\mathbf{x} + \sqrt{\beta_1}\cdot\boldsymbol{\epsilon}_1 \\
\mathbf{z}_t &= \sqrt{1-\beta_t}\cdot\mathbf{z}_{t-1} + \sqrt{\beta_t}\cdot\boldsymbol{\epsilon}_t \qquad \forall\, t \in 2,\ldots,T,
\end{aligned}$$

where ϵt is noise drawn from a standard normal distribution. The first term attenuates the data plus any noise added so far, and the second adds more noise. The hyperparameters βt ∈ [0, 1] determine how quickly the noise is blended and are collectively known as the noise schedule. The forward process can equivalently be written as:

$$\begin{aligned}
q(\mathbf{z}_1|\mathbf{x}) &= \mathrm{Norm}_{\mathbf{z}_1}\left[\sqrt{1-\beta_1}\mathbf{x}, \beta_1\mathbf{I}\right] \\
q(\mathbf{z}_t|\mathbf{z}_{t-1}) &= \mathrm{Norm}_{\mathbf{z}_t}\left[\sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t\mathbf{I}\right] \qquad \forall\, t \in 2,\ldots,T.
\end{aligned}$$

Decoder model (reverse process)

When we learn a diffusion model, we learn the reverse process. In other words, we learn a
series of probabilistic mappings back from latent variable zT to zT −1 , from zT −1 to zT −2 ,
and so on, until we reach the data x.

References

Prince, S. J. D. (2023). Understanding Deep Learning.

Elgendy, M. (2020). Deep Learning for Vision Systems.

Ravi, J. (Year). Build GANs and Diffusion Models with TensorFlow and PyTorch (Linkedin course)

Contact with me

LINKEDIN

FACEBOOK

GITHUB

KAGGLE

SEE YOU IN NEW ADVENTURE