

## 2. Autoencoder Clásico

El Autoencoder (AE) es un tipo de red NO SUPERVISADA diseñada para extraer características de los datos de entrada y representarlas en un espacio de menor dimensión (espacio latente), para después reconstruir estos datos de entrada a partir de lo extraído en el espacio latente.

### 2.1. Estructura

El Autoencoder se compone de 3 partes:

- **Encoder:** Es el encargado recibir los datos y llevarlos a su representación latente. En imágenes, utiliza redes convolucionales y Pooling para la extracción de características y la compresión de dimensión espacial. Esta transformación se representa como:

$$z = \text{Encoder}(x)$$

Donde,

$z$  = representación latente

$x$  = datos de entrada

- **Cuello de botella:** Es donde reside la representación latente de los datos (espacio latente).
- **Decoder:** Es el encargado de tomar un vector latente y realizar la reconstrucción de los datos hacia la entrada original. En imágenes, utiliza redes convolucionales transpuestas para la reconstrucción y aumento progresivo de la dimensión espacial. Esta transformación se representa como:

$$\hat{x} = \text{Decoder}(z)$$

Donde,

$z$  = representación latente

$\hat{x}$  = datos de salida

Así, se busca minimizar la diferencia entre los datos de entrada y salida, para llegar lo más cerca posible a:  $\hat{x} = x$ .

El Autoencoder clásico, tiene como propósito la reducción de dimensionalidad, eliminación de ruido, detección de anomalías, sin embargo, este modelo cuenta con ciertas limitaciones.

Las limitaciones se dan porque el espacio latente ( $z$ ) no está regularizado. No garantiza continuidad ni completitud, haciendo que la generación sea impredecible.

### 2.2. Función de pérdida

El Autoencoder se entrena minimizando una función de pérdida de reconstrucción (reconstruction loss) entre la entrada original  $x$  y la salida reconstruida  $\hat{x}$ . En el caso de imágenes, se calcula el error pixel a pixel con la métrica elegida y luego se promedia para obtener así el error total.

#### Métricas de pérdida reconstrucción

Existen dos métricas comunes utilizadas en esta arquitectura:

#### Error cuadrático medio (MSE)

$$\mathcal{L}_{MSE}(x, \hat{x}) = \frac{1}{N} \sum_{i=1}^N (x_i - \hat{x}_i)^2$$

## Entropía cruzada binaria (BCE)

$$\mathcal{L}_{BCE}(x, \hat{x}) = -\frac{1}{N} \sum_{i=1}^N (x_i \cdot \log(\hat{x}_i) + (1 - x_i) \cdot \log(1 - \hat{x}_i))$$

La diferencia clave entre estos es la forma de la función, la BCE genera gradientes más grandes, lo que se traduce en una reducción del error más rápida a través del backpropagation, sin embargo, también indica un mayor costo computacional. Además, BCE, como su nombre lo indica es para datos binarios, por tanto, para utilizarla en imágenes requiere la normalización de los píxeles.

## Nota: Backpropagation

A la hora de aplicar el backpropagation para la actualización de parámetros, se tiene que:

$$W_n = W_o - \mu \frac{\partial \mathcal{L}}{\partial W_o} \text{ (con descenso de gradiente)}$$

Donde,

$W_n$  = pesos nuevos

$W_o$  = pesos antiguos

$\mu$  = tasa de aprendizaje (hiperparámetro)

La función de reconstrucción ( $\mathcal{L}$ ) se puede derivar parcialmente respecto a los pesos ( $W$ ) porque la salida reconstruida  $\hat{x}$  depende implícitamente de estos pesos.

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \frac{\partial \mathcal{L}}{\partial \hat{x}^{(l)}} \cdot \frac{\partial \hat{x}^{(l)}}{\partial z^{(l)}} \cdot \frac{\partial z^{(l)}}{\partial W^{(l)}}$$

Donde,

$$\hat{x}^{(l)} = \sigma(z^{(l)})$$

$$z^{(l)} = W^{(l)} \cdot \hat{x}^{(l-1)} + b^{(l)}$$

Donde,

$\hat{x}^{(l)}$  = activación en la capa ( $\uparrow$ )

$\hat{x}^{(l-1)}$  = activación en la capa anterior ( $\uparrow - \infty$ )

$b^{(l)}$  = sesgo de la capa ( $\uparrow$ )

$l$  = capa de la red

$\sigma$  = función de activación (sigmoide, ReLU, tanh, etc)

Así, los términos asociados a las derivadas parciales son,

$$\frac{\partial \mathcal{L}}{\partial \hat{x}^{(l)}} = 2(\hat{x}^{(l)} - x) \text{ (para MSE)}$$

$$\frac{\partial \hat{x}^{(l)}}{\partial z^{(l)}} = \sigma'(z^{(l)})$$

$$\frac{\partial z^{(l)}}{\partial W^{(l)}} = \hat{x}^{(l-1)}$$

Por lo que el gradiente resultante es,

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = 2(\hat{x}^{(l)} - x) \cdot \sigma'(z^{(l)}) \cdot \hat{x}^{(l-1)}$$

Siendo este último el encargado de actualizar los pesos tras cada época de entrenamiento del modelo.

### 3. Autoencoder Variacional

El VAE es un modelo generativo que combina la arquitectura de un Autoencoder con la inferencia Variacional. Está diseñado para aprender una distribución de probabilidad de los datos de entrada y generar nuevas muestras.

#### 3.1. Estructura

El VAE se compone de:

- **Encoder:** El codificador en este caso, lo que hace es encontrar los parámetros de una distribución aproximada variacional  $q(z|x)$ , quien representa los datos en el espacio latente. Así, el encoder recibe una muestra  $x$  y genera la media y desviación estándar  $(\mu, \sigma)$  de la distribución en el espacio latente. Se asume que la distribución posterior  $q(z|x)$  es una Gaussiana multivariada diagonal (varianzas independientes).

$$q(z|x) = \mathcal{N}(z|\mu_\phi(x), \Sigma_\phi(x))$$

- **Cuello de botella:** En esta parte de la estructura, se aplica el llamado truco de reparametrización, y es que, la operación de muestreo  $z \sim q(z|x)$  no es diferenciable, lo que impide realizar el backpropagation. La solución es calcular la variable latente de forma determinista, tomando una variable aleatoria auxiliar ( $\epsilon$ ) extraída de una gaussiana estándar. Luego, se escala multiplicando por la desviación estándar y se desplaza por la media.

$$z = \sigma \odot \epsilon + \mu$$

$\odot$  : producto elemento a elemento

Al incluir esta variable externa, permite realizar el backpropagation, ya que la única variable aleatoria ( $\epsilon$ ) no depende de los pesos, entonces al separar la variable aleatoria, los pesos influyen de manera determinista en la muestra.

**Nota**  $z \sim q(z|x)$  no es diferenciable porque es una operación estocástica, es el muestreo de  $z$  sobre la distribución  $q(z|x)$ .

- **Decoder:** El decodificador utiliza la verosimilitud  $p(x|z)$ , aquí, recibe la muestra  $z$  y la transforma en la reconstrucción  $\hat{x}$ . En otras palabras, modela la probabilidad de generar la muestra original  $x$  dada la muestra latente  $z$ .
- **Función de pérdida:** El entrenamiento maximiza el ELBO, midiendo así la fidelidad de la reconstrucción de los datos.

### 4. Autoencoder Variacional Condicionado

Un CVAE es una variación del VAE que permite añadir información adicional al modelo (etiquetas, parámetros, etc.) para guiar la inferencia latente, así como la generación/estimación de nuevas muestras. A la información adicional se le llama “condición” y se va a denotar como  $\mathbf{y}$ .

El objetivo del CVAE es modelar la distribución original de los datos dado el condicionamiento, es decir:

$$p(x|y)$$

Donde  $\mathbf{y}$  es una variable externa que influye directamente en los datos. De esta manera, se evita que el VAE sea quien capture e interprete todas las características por sí mismo, haciendo que las distribuciones del modelo dependan explícitamente de la condición.

En este modelo, los términos pasan a ser condicionados por  $\mathbf{y}$  y se escriben como:

$$p(x|y) = \int p(x|z, y)p(z|y)dz$$

Donde,

- $p(x|y)$  posterior condicionada
- $p(z|y)$  es el prior condicionado
- $p(x|z, y)$  es la verosimilitud condicionada (decoder)

Entonces, el encoder ahora debe aproximar la distribucion marginal condicionada:

$$p(z|x, y) \approx q(z|x, y)$$

Como la condición es una variable independiente, se toma como una constante, entonces el tratamiento para llegar al ELBO es el mismo:

$$\mathcal{L}_{CVAE} = -\mathbb{E}_{q(z|x, y)} [\log (p(x|z, y))] + \mathbb{D}_{\text{KL}}((q(z|x, y)||p(z, y)))$$

#### 4.1. Estructura

La estructura general es la misma a la de un VAE, la diferencia es que el condicionamiento se hace en ambas partes del modelo. Este condicionamiento se puede hacer de distintas maneras (concatenación, FiLM, cross-attention, embeddings), según el método que se elija va a variar el código, pero, en general, así se describe la estructura de cualquiera de estas implementaciones:

- Encoder: Recibe la imagen y la condición, con esto produce los parámetros de la distribución latente:

$$q(z|x, y) = \mathcal{N}(z|\mu(x, y), \Sigma(x, y))$$

El encoder “decide” dónde ubicar los datos en el espacio latente teniendo en cuenta la condición. Esto fuerza a que los datos con condiciones similares ocupen regiones latentes coherentes.

- Cuello de botella: Se aplica de igual manera el truco de la reparametrización:

$$z = \sigma \odot \epsilon + \mu$$

- Prior condicionado: Se cambia el prior de una distribución normal por:

$$p(z|y) = \mathcal{N}(\mu(y), \sigma(y)I)$$

Donde  $\mu(y)$  y  $\sigma^2(y)$  son parámetros que se obtienen de la condición.

- Decoder: Toma como entrada el vector latente junto con la condición y genera la reconstrucción condicionada:

$$p(x|z, y)$$

De manera más explícita, se describe cómo lo hace cada variante de los CVAE.

#### 4.2. Concatenación

Consiste en adjuntar (concatenar) directamente la condición  $y$  al tensor de entrada o al vector latente. Es la forma más simple de introducir el condicionamiento.

**Encoder:** Se concatena la imagen con la condición a lo largo del canal o de la dimensión de características. Luego, el encoder procesa esta entrada combinada para obtener los parámetros de la distribución variacional:

$$q(z|x, y) = \mathcal{N}(z|\mu(x, y), \Sigma(x, y))$$

**Decoder:** El decoder recibe el vector latente  $z$  concatenado nuevamente con la condición  $y$ :

$$p(x|z, y)$$

Ventajas	Desventajas
<ul style="list-style-type: none"> <li>• Fácil implementación.</li> <li>• Funciona bien para etiquetas simples (ej. etiquetas distretas)</li> </ul>	<ul style="list-style-type: none"> <li>• Si la condición tiene mucha estructura (un texto, parámetros continuos complejos, etc), no suele ser suficiente para alinear correctamente las representaciones.</li> </ul>

Cuadro 3: Concatenación: ventajas vs desventajas

### 4.3. Embedding semántico (prior)

En este método, la condición se transforma para obtener un vector embebido (embedding), quien es el que captura la información semántica de la condición.

**Encoder:** El encoder permanece sin modificaciones, eso se hace con el fin de evitar sobre ajustes al darle la condición de entrada.

**Embedding de la condición** La condición se pasa por una red de embebido para la extracción de la información semántica:

$$e_y = \text{Embedding}(y)$$

**Decoder:** El decoder recibe el vector latente  $z$  junto con el embedding  $e_y$ , lo que permite la reconstrucción de una imagen coherente en función del condicionamiento. Así, el prior resultante es:

$$p(z|y) = \mathcal{N}(\mu_y, \sigma_y)$$

Donde  $\mu_y$  y  $\sigma_y$  se obtienen del embebido  $e_y$

*Nota: con información semántica se hace referencia a la parte del dato que expresa su significado, contexto o intención, no solo su forma numérica o superficial.*

Ventajas	Desventajas
<ul style="list-style-type: none"> <li>• Convierte etiquetas, textos o parámetros en representaciones ricas en semántica.</li> <li>• Permite entrenar modelos con condiciones complejas.</li> </ul>	<ul style="list-style-type: none"> <li>• Requiere diseñar un embedding adecuado.</li> <li>• Introduce de parámetros adicionales.</li> </ul>

Cuadro 4: Embedding: ventajas vs desventajas

### 4.4. FiLM

Este método toma la condición y modula las características intermedias mediante transformaciones lineales por canal. Así, para cada capa del encoder/decoder, se aprenden dos funciones de la condición:

$$\gamma(y), \beta(y)$$

Estas modulan de forma de los mapas de características:

$$\text{FiLM}(y) = \gamma(y) \odot h + \beta(y)$$

Donde  $h$  son las activaciones de la red.

**Encoder:** Las características intermedias se modifican en función de la condición, haciendo que el encoder aprenda cómo la condición afecta la posición de los datos en el espacio latente.

**Decoder:** La generación también se modula apa por capa, generando así las imagenes condicionadas.

Ventajas	Desventajas
<ul style="list-style-type: none"><li>• Buen acondicionamiento.</li><li>• Control del flujo de información.</li><li>• Efectivo con parámetros continuos</li></ul>	<ul style="list-style-type: none"><li>• Implementación compleja.</li><li>• Introducción de nuevas capas para calcular <math>\gamma(y)</math> y <math>\beta(y)</math>.</li></ul>

Cuadro 5: FiLM: ventajas vs desventajas